ENKCC-1.2 VAX 11/730 MCT MICRO

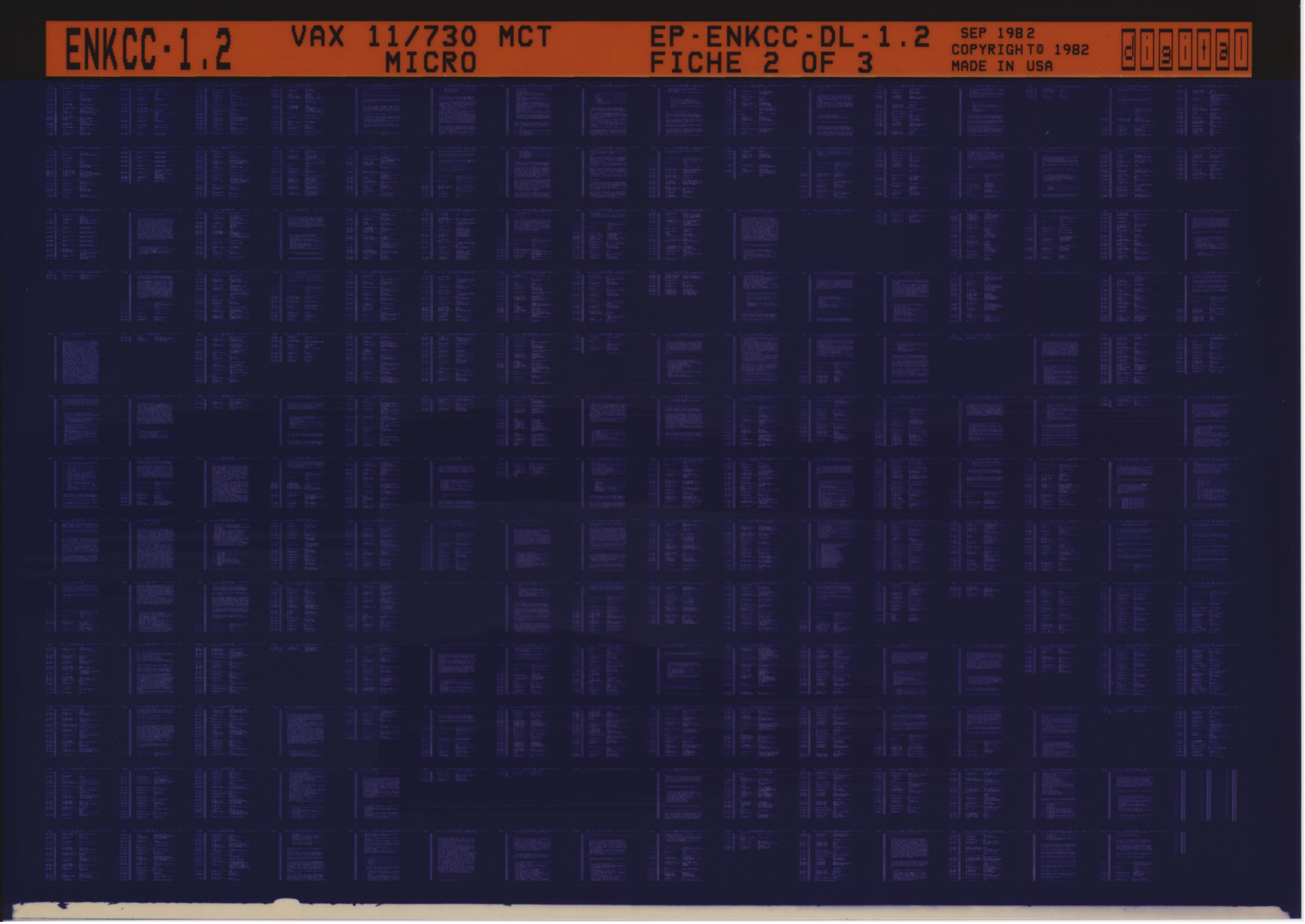
EP-ENKCC-DL-1.2 FICHE 1 OF 3

SEP 1982
COPYRIGHT© 1982
MADE IN USA

digital

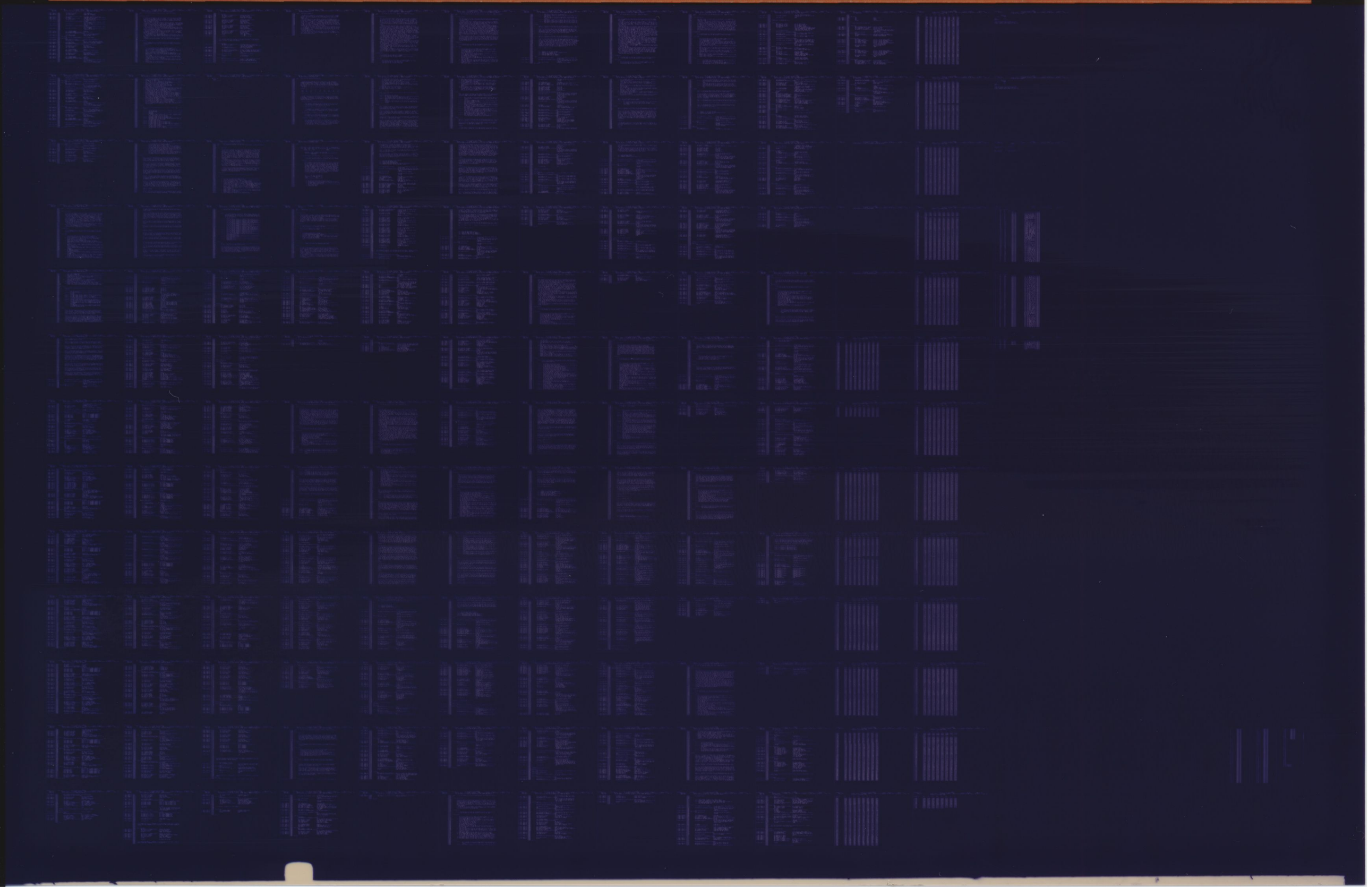
ENKCC-1.2 VAX 11/730 MCT MICRO EP-ENKCC-DL-1.2 FICHE 2 OF 3

SEP 1982
COPYRIGHT© 1982
MADE IN USA



ENKCC-1.2 VAX 11/730 MCT MICRO EP-ENKCC-DL-1.2 FICHE 3 OF 3

SEP 1982
COPYRIGHT© 1982
MADE IN USA



IDENTIFICATION

Product code: ZZ-ENKCC VERSION 1.2

Product name: MICRO-DIAGNOSTIC FOR VAX-11/730 MCT MODULE

Product date: 8-MARCH-82

Maintainer: BASE SYSTEMS DIAGNOSTIC ENGINEERING

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital or its affiliated companies.

Copyright (c) 1982 by Digital Equipment Corporation. All Rights Reserved.

The following are trademarks of Digital Equipment Corporation.

DEC	DECsystem-10	DECSYSTEM-20
DECUS	MASSBUS	PDP
UNIBUS	VAX	VMS

digital

Table of Contents

1.0 ABSTRACT 3

2.0 HARDWARE REQUIREMENTS 3

3.0 SOFTWARE REQUIREMENTS 3

4.0 PREREQUISITES 3

5.0 OPERATING INSTRUCTIONS 4
5.1 Event Flags 4
5.2 APT Setup 4

6.0 PROGRAM FUNCTIONAL DESCRIPTION 5
6.1 Program Overview 5
6.2 Program Size 5
6.3 Program Run Times 5
6.4 Fault Detection 5
6.5 Performance During Hardware Failures 6
6.6 Program Applications 6
6.7 Test Descriptions 6

7.0 MAINTENANCE HISTORY 7

1.0 ABSTRACT

This program is a micro-diagnostic designed to test the hardware on the MCT (memory controller) module of the VAX-11/730 processor. In addition, it will test all main memory arrays found in the system. If a UBE is present, the test will also check additional UNIBUS logic.

2.0 HARDWARE REQUIREMENTS

This program will run with the minimum hardware configuration of a WCS, CPU, MCT, and array module. If a UBE is installed and configured as described in the program listing, the program will run additional UNIBUS logic tests automatically.

Maximum main memory allowed is 5 one meg boards, for a total of 5 megabytes of main memory storage.

Other options may be installed without affecting the diagnostic's performance.

3.0 SOFTWARE REQUIREMENTS

This diagnostic must be run under the VAX-11/730 micro monitor (ENKAA) in a standalone environment. The micro-diagnostic is written into WCS RAM, wiping out any system micro-code that may have been resident. It will also use all of main memory during testing. The micro-diagnostic monitor takes up the area where the console software is normally present.

4.0 PREREQUISITES

The WCS and DAP modules are assumed to have been tested and known to be good before this diagnostic is run. The programs ENKBA through ENKBE, ENKCA, and ENKCB will perform this testing.

5.0 OPERATING INSTRUCTIONS

This program may be executed by typing DI SE ENKCC after the micro-monitor has been loaded and the MIC> prompt has been printed at the terminal. Both the micro-monitor and this program are loaded from a TU58 cassette or downline loaded through the APT-RD port. See the OVERVIEW MANUAL for more information.

5.1 Event Flags

Not applicable

5.2 APT Setup

The micro-monitor knows when APT is present by the position of the keyswitch and the state of a line connected to the APT-RD port. It sets a flag in the CPU Local Store which can be read by this diagnostic. When APT is present, the diagnostic will:

1. compare the result of it's memory sizing routine with a parameter supplied by APT and stored by the monitor in the CPU Local Store. If the results are not equal, a standard error message is issued.

2. Report any single bit errors as a standard error rather than reporting only the number of single bit errors found.

Under APT, the diagnostic will halt on error and report error information to APT via the mailbox in 8085 RAM.

While creating a program to run this diagnostic under APT, the following parameters must be specified:

MEMORY SIZE IN 1/4 MB BLOCKS: Enter the size of memory in 1/4 MB increments. For example, enter "4" if the machine has 1 MB of memory.

HARDWARE BYTE: Enter 0 if no UBE is present, 1 if UBE is present.

SOFTWARE BYTE: Enter 0, (Default), Moving Inversion Test, (Test 33) enabled, 1 Test 33 disabled. Inclusion of this test adds a little less than 3 minutes to the run time of ENKCC for each meg of memory.

6.0 PROGRAM FUNCTIONAL DESCRIPTION

6.1 Program Overview

This program is designed to detect stuck-at faults and provide a repair tool which helps isolate the failing component. A bottom up diagnostic strategy is utilized which builds on previous tests. The tests should be run in consecutive order, to gain the best isolation of a failing component.

6.2 Program Size

This program runs in WCS RAM memory and is approximately 26K bytes long (utilizing about 8500 micro-words in WCS).

6.3 Program Run Times

This program will take approximately 4 minutes to run on a system with 1 megabyte of memory and no UBE installed, exclusive of load time. If a UBE is installed, an additional few seconds will be added to the run time.

This program will take about 3 1/2 minutes longer to run for each additional 1 megabyte of main memory on the system after the first megabyte.

6.4 Fault Detection

The diagnostic will report errors with the following header:

```
SECT  TST ERR   EXP   REC  OTHER  MSK  MODULE
```

1. SECT is the program name (ENKCC)
2. TST is the test number that failed.
3. ERR is the error number that failed.
4. EXP is the expected (correct) data.
5. REC is the received (actual) data.
6. OTHER is any other pertinent data (such as an address). This field is not always used. N/A will be printed under OTHER when not in use. The ERRORS section in the listing will describe the contents of this field when it is used.

7. MASK is the error mask used to check the result. Bits set to a 1 in this mask correspond to bits in the result that are not checked.

8. MODULE is the suspected module that caused the failure.

6.5 Performance During Hardware Failures

A power fail will cause a rebooting of the system. Other failures, such as a timeout or a WCS parity error, will print an error message and return to the MIC> prompt. The operator can then issue other commands, including a continue if desired.

6.6 Program Applications

This program can be used by field service to determine which module should be replaced and to verify that the replacement eliminated the failure. It can also be used as a repair tool to help isolate a failing component by manufacturing, field service or engineering. The program is APT compatable.

Customers may used the program to verify the basic integrity of the memory controller and array modules in the system.

6.7 Test Descriptions

See the actual test in the listing for detailed descriptions and error analyses. A brief description of the logic being tested by each test can be found in the table of contents of the listing.

7.0 MAINTENANCE HISTORY

DATE	VERSION	DESCRIPTION
8-MAR-82	1.0	Initial release.
18-MAY-82	1.1	Test 44, (Moving Inversion Test), moved to Test 33 and made modifications to make it a default test. Changes were made to enable or disable Test 33 via Software Byte under APT. Also a fix was made to run APT without a U.B.E. board.
10-JUN-82	1.2	Test 42, changes made to allow a variable timing loop to ensure UBE to memory transfers weren't so critical.

Table of Contents

FIELD DEFINITIONS FOR MAIN MICRO WORD VER 00.08
MACRO DEFINITIONS VER 00.02
1719 FIELD DEFINTITIONS FOR DATA PATH CONTROL MICRO WORD VER 00.01
1772 CONTROL ROM CONTENTS VER 00.01
2521 DIAGNOSTIC MACROS AND DEFINITIONS
3180 COMMON LS ASSIGNMENTS (TO REGISTERS)
3206 PROGRAM SPECIFIC LS ASSIGNMENTS (LS 80-F7)
3259 Microinstruction Formats
3642 Tables
3773 2901 ALU Control Description
3876 DIAGNOSTIC MACROS
3901 TEST POINTERS
4003 DIAGNOSTIC POINTERS
4092 LS DATA SECTION
4483 PROGRAM SPECIFIC DATA SECTION (LS 80-F7)
4858 WCS SUBROUTINES FOR CONSOLE USE
4859 GENERATE TRANSFER DATA
4904 DEPOSIT MCT CSR ROUTINE
4970 EXAMINE MCT CSR ROUTINE
5057 DEPOSIT MCT TRANSLATION BUFFER ROUTINE
5173 EXAMINE TRANSLATION BUFFER ROUTINE
5236 DEPOSIT UNIBUS MAP ROUTINE
5334 EXAMINE UNIBUS MAP ROUTINE
5397 DEPOSIT MAIN MEMORY ROUTINE
5455 EXAMINE MAIN MEMORY ROUTINE
5517 EXAMINE IDC ROUTINES
5583 DEPOSIT IDC ROUTINES
5655 SAVE WR ROUTINE
5706 RESTORE WR ROUTINE
5757 WCS SUBROUTINES FOR CPU USE
5758 ERROR ROUTINE
5925 START TRANSFER ROUTINE
6020 MCT CSR BIT ALLOCATIONS
6084 Memory Initialization
6100 Memory Dispatch Description
6132 Micro Tests
6133 VAR Tests
6135 TEST 1 Write/read VAR Test (M8391 MCT Module and M8390 CPU module)
6411 TEST 2 Inc VAR and Stall On Mem Busy Test (M8390 CPU and M8391 MCT Module)
6661 Basic CSR 1 Test and Limited Branch tests
6663 TEST 3 CSR 1 (partial) Read/Write Test (M8391 MCT and M8390 CPU Module)
6872 TEST 4 Branch on LVA00, ECC DIS, and DIAG CHECK Test (M8391 MCT Module)
7095 UNIBUS address, data and control logic tests
7097 TEST 5 UNIBUS Address Test (M8391 MCT Module)
7261 TEST 6 UNIBUS Data Test (M8391 MCT Module and M8394 WCS Module)
7413 TEST 7 Check MOV MEM.DATA TO LS Logic (M8390 CPU Module)
7507 TEST 8 Branch On UBC1,IC0, UB ACTIVITY Test (M8391 MCT Module)
7746 TEST 9 MYSN and SSYN Branch Test (M8391 MCT Module)
7897 Data Rotator and ECC Tests
7899 TEST A Write/read Data Through Rotator PAL (M8391 MCT Module)
8111 TEST B ARY BUS, ECC Latches, and CSR CKBTS Test (M8391 MCT Module)
8418 TEST C ECC Checkbit Generation Test (M8391 MCT Module)
8719 TEST D ECC Error (no correction) Test (M8391 MCT Module M8390 CPU Module)
9222 TEST E ECC Correction Test (M8391 MCT Module)
9812 Data Rotation Tests

Table of Contents

; 9814 TEST F Rotate 9 Bits Right (M8391 MCT Module)
; 9929 TEST 10 Rotate 15 Bits Left (M8391 MCT Module)
; 10037 Translation Buffer, Unibus Map, and TAG Tests
; 10039 TEST 11 TB RAM Test (M8391 MCT Module)
; 10656 TEST 12 UNIBUS Map RAM Test (M8391 MCT Module)
; 11151 TEST 13 TB Step Test (M8391 MCT Module)
; 11292 TEST 14 TBA 09 and TB/UBS March Test (M8391 MCT Module)
; 11519 TEST 15 TB Miss and Tag Field Test (M8391 MCT Module)
; 12270 TB Parity And CSR 1 Valid Bit Tests
; 12272 TEST 16 TB Parity Test Including TB P0 Bit of TB RAM (M8391 MCT Module)
; 12672 TEST 17 CSR 1 VALID Bit Test (M8391 MCT Module)
; 12783 Protection Prom Tests
; 12785 TEST 18 TEST.V.WCHK Test (for use in next test) (M8391 MCT Module)
; 12906 TEST 19 Protection Prom CSR 1 Access/Modify Refused Test (M8391 MCT Module)
; 13312 Other CSR 1 Bits Test
; 13314 TEST 1A NXM And Adapter Reg Select Test (M8391 MCT Module)
; 13539 TEST 1B ILL UB OPER (CSR bit 20) Test (M8391 MCT or M8390 CPU Module)
; 13834 TEST 1C Write Across Page Error Bit (Bit 19) (M8391 MCT Module)
; 14166 Main Memory Array Tests
; 14168 TEST 1D Size and Initialize Main Memory (M8391 MCT or M8728 ARRAY Module)
; 14411 TEST 1E All 1's and All 0's Data Test (M8728 ARRAY Module, M8391 MCT Module)
; 14678 TEST 1F Data Path Test (M8728 ARRAY)
; 14788 TEST 20 Refresh Test (M8728 ARRAY, M8394 WCS Module)
; 14987 TEST 21 Main Memory March Test (M8391 MCT or M8728 ARRAY Module)
; 15558 Other Memory Controller Logic
; 15560 TEST 22 NXM On Access To Non-Existant Unibus Device (M8391 MCT Module)
; 15671 TEST 23 Abort Instruction On CS Parity Error (M8391 MCT or M8390 CPU Module)
; 15832 TEST 24 March Test of CSR 1 Bits and ERROR SUM (M8391 MCT Module)
; 16342 TEST 25 Byte and Word Write To Memory (M8391 MCT Module)
; 16662 TEST 26 MME Test (M8391 MCT Module)
; 16822 TEST 27 Prefetch Logic (M8391 MCT or M8390 CPU Module)
; 17102 TEST 28 Abort IB Fill on Uncorrectable Error (M8391 MCT Module)
; 17282 TEST 29 Clear CSR and Init On CINIT, UBS DCLO (M8391 MCT or M8394 WCS Module)
; 17516 Micro-flow Verification Tests
; 17517 TEST 2A Read Array Micro-flow Test (M8391 MCT Module)
; 17627 TEST 2B Read Array Flow with Single/Double Bit Error (M8391 MCT Module)
; 17933 TEST 2C DIAG CHK and ECC DISABLE Mode Tests (M8391 MCT Module)
; 18383 TEST 2D READ.V Tests Without Unibus Activity (M8391 MCT Module)
; 18608 TEST 2E READ.V.RCHK.IFILL Flow with Error (M8391 MCT Module)
; 18808 TEST 2F WRITE.V.WCHK Flow and Abort Write (M8391 MCT Module)
; 18961 TEST 30 WRITE ARRAY Flow with ECC DIS or ERROR (M8391 MCT Module)
; 19424 TEST 31 WRITE OCTA Flow (M8391 MCT Module)
; 19951 TEST 32 READ QUAD and READ OCTA Flow (M8391 MCT Module)
; 20724 Extensive Array and Memory Timing Test
; 20725 TEST 33 Moving Inversion Test on Array (M8728 ARRAY or M8391 MCT Module)
; 21297 UBE MICRO DIAGNOSTICS
; 21322 UBE SETUP AND REGISTER DESCRIPTION
; 21421 UBE Tests
; 21422 TEST 34 UBE Present Test
; 21540 TEST 35 Write/Read UBE Registers and DC LO Test (M8391 MCT Module)
; 21787 TEST 36 Verify Byte Access on Read to UNIBUS (M8391 MCT Module)
; 21888 TEST 37 BR Interrupt Test (M8391 MCT or M8390 CPU Module)
; 22178 TEST 38 UBE Write To Array Test (M8391 MCT Module)
; 22513 TEST 39 UNIBUS DATOB And TWO MEM CYC Write to Array Test (M8391 MCT Module)
; 22890 TEST 3A UNIBUS DATO With CRD and RDS Error Test (M8391 MCT Module)

Table of Contents

; 23152 TEST 3B UNIBUS DATO Crossing Page Boundary Test (M8391 MCT Module)
; 23391 TEST 3C Partial UB CSR 2 and UB ERRSUM Test (M8391 MCT Module)
; 23881 TEST 3D UBE Read To Array Test (M8391 MCT Module)
; 24111 TEST 3E UNIBUS DATI With CRD and RDS Error Test (M8391 MCT Module)
; 24506 TEST 3F UB CSR 2 and UB ERRSUM On DATI Test (M8391 MCT Module)
; 24760 TEST 40 ISSUE BG Sack Timeout Test (M8391 MCT Module)
; 24828 TEST 41 Block Transfer Test (M8391 MCT Module)
; 24957 TEST 42 LOCK and UB BSY Logic Test (M8391 MCT Module)
; 25308 Test 43 UBE Multiple Write/Read To Array Test (M8391 MCT Module)
; 25487 Read UBE Regs Routine (Debug routine. Only run on DI TE 44)
; 25556 MCT SUBROUTINES

```
                ;1      .TITLE  'ENKCC Micro-diagnostic for MCT module  Rev 01.2'
                ;2      ;
                ;3      ;
                ;4      ;                       COPYRIGHT (c) 1982 BY
                ;5      ;               DIGITAL EQUIPMENT CORPORATION, MAYNARD,
                ;6      ;                MASSACHUSETTS.  ALL RIGHTS RESERVED.
                ;7      ;
                ;8      ; THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY  BE  USED  AND  COPIED
                ;9      ; ONLY  IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE INCLUSION
                ;10     ; OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE OR ANY OTHER COPIES THEREOF
                ;11     ; MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY OTHER  PERSON.  NO
                ;12     ; TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
                ;13     ;
                ;14     ; THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE, AND
                ;15     ; SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
                ;16     ;
                ;17     ; DIGITAL  ASSUMES  NO  RESPONSIBILITY  FOR  THE  USE OR RELIABILITY OF ITS
                ;18     ; SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.
                ;19     ;
                ;20
                ;21     ;++
                ;22     ; FACILITY:     VAX-11/730 MICRO-DIAGNOSTIC.
                ;23     ;
                ;24     ; ABSTRACT:     This micro-diagnostic tests the hardware of the MCT module
                ;25     ;               in the VAX-11/730 processor.
                ;26     ;
                ;27     ; ENVIRONMENT:  ENKAA  Micro-diagnostic Monitor
                ;28     ;
                ;29     ; AUTHOR:       Ernest Preisig          26-AUG-81
                ;30     ;
                ;31     ; MODIFIED BY:  David Mayo              8-MAR-82        VERSION 01.00
                ;32     ;               Initial release.
                ;33     ;
                ;34     ;               Ernie Preisig           29-Apr-82       VERSION 01.10
                ;35     ;               Peter Green
                ;36     ;               Modified test 44 and made it test 33
                ;37     ;               Fixed UBE present test under APT
                ;38     ;
                ;39     ;               Ernie Preisig           9-Jun-82        VERSION 01.20
                ;40     ;               Peter Green
                ;41     ;               Modified test 42 to allow a variable
                ;42     ;               timing loop for memory to unibus
                ;43     ;               due to UBE inconsistancies.
                ;44     ;
                ;45     ;--
                ;46     .nolist
                ;50     .list
                ;51     .NOCREF
;52             .NOBIN
;53             .SEQUENTIAL
;54             .RTOL
;55             .HEXADECIMAL
```

```
;56      .PAGE
;57      .TOC    "FIELD DEFINITIONS FOR MAIN MICRO WORD                  VER 00.08"
;58      .SET/UPC=<000>
;59
;60      ; THE FOLLOWING EXPRESSIONS ARE VALIDITY CHECKS FOR FIELD COMBINATIONS
;61
;62      .SET/A.VAL=<.CASE[<OPC2/>]OF[0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0]>
;63      .SET/B.VAL=<.CASE[<OPC2/>]OF[0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1]>
;64      .SET/D.VAL=<.CASE[<OPC2/>]OF[0,0,0,1,0,0,0,0,1,1,1,1,1,1,1,1]>
;65      .SET/XD.VAL=<.EQL[<OPC1/>,<OPC1/MOVE>]>
;66      .SET/CC.VAL=<.CASE[<OPC2/>]OF[0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1]>
;67      .SET/DT.VAL=<.EQL[<OPC2/>,<OPC2/MEM.REQ>]>
;68      .SET/SCTL.VAL=<.CASE[<OPC2/>]OF[0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1]>
;69      .SET/JCTL.VAL=<.CASE[<OPC2/>]OF[1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]>
;70      .SET/JUMP.VAL=<.EQL[<OPC2/>,<OPC2/JUMP>]>
;71      .SET/DECODE.VAL=<.EQL[<OPC2/>,<OPC2/DECODE>]>
;72      .SET/MISC.VAL=<.EQL[<OPC2/>,<OPC2/MISC>]>
;73      .SET/DP.VAL=<.EQL[<OPC/>,<OPC/BASIC>]>
;74      ;
;75      ;       THE FOLLOWING VALIDITY CHECKS ARE USE FOR THE PAGE BOUNDARY PROBLEM.
;76      ;
;77      .SET/PAGE.PROB=<.EQL[<.AND[<7FF>,<.>]>,<7FE>]>
;78      .SET/SKIP.LEGAL2=<.OR[<.EQL[<SCTL/>,<SCTL/RET.NOERR>]>,<.EQL[<SCTL/>,
;79      <SCTL/POP.USTACK>]>]>
;80      .SET/SKIP.LEGAL=<.OR[<.EQL[<SCTL/>,<SCTL/NO.SKIP>]>,<.EQL[<SCTL/>,
;81      <SCTL/RETURN>]>,<.EQL[<SCTL/>,<SCTL/RETURN+1>]>,<SKIP.LEGAL2>]>
;82      .SET/SKIP.PAGE.VAL=<.CASE[<PAGE.PROB>]OF[1,<SKIP.LEGAL>]>
;83      .SET/JUMP.CHECK=<.OR[<.EQL[<JCTL/>,<JCTL/JSR>]>,<.EQL[<JCTL/>,
;84      <JCTL/JSR.VALID>]>]>
;85      .SET/JUMP.PAGE.VAL=<.CASE[<PAGE.PROB>]OF[1,<.XOR[1,<JUMP.CHECK>]>]>
;86
;87
;88      ; MICRO INSTRUCTION OPCODE DEFINITIONS
;89      ;
;90      OPC/=<22>,.DEFAULT=<OPC/BASIC>  ; MICRO OPCODE FIELD FOR SINGLE BIT OPCODE.
;91
;92              BASIC=1                 ; OPCODE FOR 'BASIC' MICRO INSTRUCTION
;93
;94      OPC1/=<22:20>                   ; MICRO OPCODE FIELD FOR THREE BIT OPCODES.
;95
;96              EXTENDED=2              ; OPCODE FOR 'EXTENDED' MICRO INSTRUCTION
;97              MOVE=3                  ; OPCODE FOR 'MOVE' MICRO INSTRUCTION
;98
;99      OPC2/=<22:19>                   ; MICRO OPCODE FIELD FOR FOUR BIT OPCODES
;100
;101             MEM.REQ=3               ; OPCODE FOR 'MEM.REQ' MICRO INSTRUCTION
;102             MISC=2                  ; OPCODE FOR 'MISC' MICRO INSTRUCTION
;103             JUMP=1                  ; OPCODE FOR 'JUMP' MICRO INSTRUCTION
;104             DECODE=0                ; OPCODE FOR 'DECODE' MICRO INSTRUCTION
;105
;106
;107     ; THE FOLLOWING FOUR FIELDS ARE RELATED TO ADDRESSING
;108     ; VARIOUS RAM MEMORIES WITHIN THE DATA PATH.
;109     ;
;110     ; THE A.ADRS  REFERS TO THE 2901 INTERNAL RAM 'A-PORT'
```

```
;111     ; THE B.ADRS  REFERS TO THE 2901 INTERNAL RAM 'B-PORT'
;112     ; THE XB.ADRS REFERS TO THE 2901 INTERNAL RAM 'B-PORT'
;113     ; THE D.ADRS  REFERS TO THE LOWER 128 LOCATIONS OF THE LOCAL STORE RAM
;114     ; THE XD.ADRS REFERS TO   ALL     256 LOCATIONS OF THE LOCAL STORE RAM
;115     ;
;116     ; THE VALIDITY STATEMENTS ALLOW USAGE OF THESE FIELDS IN THE FOLLOWING MANNER
;117     ;
;118     ;       A.ADRS          EXTENDED  MICRO INSTRUCTION
;119     ;
;120     ;       B.ADRS          BASIC     MICRO INSTRUCTION
;121     ;                       EXTENDED  MICRO INSTRUCTION
;122     ;                       MOVE      MICRO INSTRUCTION
;123     ;                       DECODE.IS MICRO INSTRUCTION
;124     ;
;125     ;       XB.ADRS         MOVE XWR  MICRO INSTRUCTION
;126     ;
;127     ;       D.ADRS          BASIC     MICRO INSTRUCTION
;128     ;                       MEM.REQ   MICRO INSTRUCTION
;129     ;
;130     ;       XD.ADRS         MOVE      MICRO INSTRUCTION
;131     ;
;132     A.ADRS/=<10:9>,.VALIDITY=<A.VAL>
;133
;134             MASK=3                  ; REGISTER BACKUP MASK
;135
;136     B.ADRS/=<8:7>,.VALIDITY=<B.VAL>,.DEFAULT=0
;137
;138             MASK=3                  ; REGISTER BACKUP MASK
;139
;140     ; THIS ADDRESS FIELD WILL CONTAIN THE LOW TWO BITS OF THE 2901 B ADDRESS
;141     ; THE THIRD BIT WILL BE FORCED BY HARDWARE.
;142     ;
;143     XB.ADRS/=<8:7>,.VALIDITY=<.EQL[<OPC1/>,<OPC1/MOVE>]>
;144
;145             BPC=0                   ; ADDRESS OF BEGINNING PC     WR[4]
;146             VA=1                    ; ADDRESS OF MEMORY REFERENCE WR[5]
;147             VAR=1                   ; ADDRESS OF LAST MEMORY REFERENCE. (WR[5] IN 2901).
;148
;149     ;D.ADRS/=<15:9>,.VALIDITY=<D.VAL>,.DEFAULT=0
;150
;151
;152     ;       SAVED.2ND.REQUEST=0     ; MM SAVED STATE FOR REQUEST.
;153     ;       T1=1                    ; TEMP LOCATION 1
;154     ;       T2=2                    ; TEMP LOCATION 2
;155     ;       T3=3                    ; TEMP LOCATION 3
;156     ;       T4=4                    ; TEMP LOCATION 4
;157     ;       T5=5                    ; TEMP LOCATION 5
;158     ;       T6=6                    ; TEMP LOCATION 6
;159     ;       T7=7                    ; TEMP LOCATION 7
;160     ;       T8=8                    ; TEMP LOCATION 8
;161     ;       T9=9                    ; TEMP LOCATION 9
;162     ;       T10=0A                  ; TEMP LOCATION 10
;163     ;       BACKUP.CM.PC=0A         ;COMPATIBILITY MODE KEEPS ORIGINAL PC HERE.
;164     ;
;165     ;       T11=0B                  ; TEMP LOCATION 11
```

```
:166    ;       T12=0C                  ; TEMP LOCATION 12                                                      ;8
:167    ;       T13=0D                  ; TEMP LOCATION 13                                                      ;8
:168    ;       T14=0E                  ; TEMP LOCATION 14                                                      ;8
:169    ;       T15=0F                  ; TEMP LOCATION 15                                                      ;8
:170    ;       PC=10                   ; MACRO PROGRAM COUNTER                                                 ;8
:171    ;       WR.BACKUP=11            ; BACKUP WORKING REGISTER FOR MOV MEM.DATA TO WR[]                      ;8
:172    ;       SAVED.VAR=12            ; MM SAVED STATE FOR VAR                                                ;8
:173    ;       SAVED.DATA=13           ; MM SAVED STATE FOR DATA                                               ;8
:174    ;       SAVED.PTE.ADDRESS=14    ; MM SAVED STATE FOR PAGE TABLE ENTRIES ADDRESS                         ;8
:175    ;       SAVED.PTE=15            ; MM SAVED STATE FOR PAGE TABLE ENTRIES                                 ;8
:176    ;       T16=16                  ; TEMP LOCATION 16                                                      ;8
:177    ;       T17=17                  ; TEMP LOCATION 17                                                      ;8
:178    ;       IE.TEMP4=18             ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION FOUR.                    ;8
:179    ;       #FFFFF000=19                    ; CONSTANT                                                      ;8
:180    ;       #FFFFFF00=1A                    ; CONSTANT                                                      ;8
:181    ;       #FFFFFFFC=1B                    ; CONSTANT                                                      ;8
:182    ;       #FFFFFFFF=1C                    ; CONSTANT                                                      ;8
:183    ;                                                                                                       ;8
:184    ;       THIS LOCATION CONTAINS THE SOFT COPY OF THE PSL.  WHENEVER THE                                  ;8
:185    ;       HARDWARE VERSION OF THE PSL IS CHANGED (PSL.HW), THEN THIS                                      ;8
:186    ;       CHANGE IS ALSO REPORTED HERE BY THE MICROCODE.  ALL BITS OF                                     ;8
:187    ;       THE PSL ARE REALLY LIKE THOSE IN THIS WORD EXCEPT FOR THE                                       ;8
:188    ;       CONDITION CODES WHICH MUST BE OBTAINED FROM PSL.CC                                              ;8
:189    ;                                                                                                       ;8
:190    ;       PSL=1D                  ; SOFT COPY OF VAX PSL                                                  ;8
:191    ;       VAR=1E                  ; VIRTUAL ADDRESS REGISTER                                              ;8
:192    ;       VAR2=1F                 ; VIRTUAL ADDRESS REGISTER                                              ;8
:193    ;                                                                                                       ;8
:194    ;                                                                                                       ;8
:195    ;       #1=20                   ; MASK 00000001 (HEX)                                                   ;8
:196    ;       #2=21                   ; MASK 00000002                                                         ;8
:197    ;       #4=22                   ; MASK 00000004                                                         ;8
:198    ;       #8=23                   ; MASK 00000008                                                         ;8
:199    ;       #10=24                  ; MASK 00000010                                                         ;8
:200    ;       #20=25                  ; MASK 00000020                                                         ;8
:201    ;       #40=26                  ; MASK 00000040                                                         ;8
:202    ;       #80=27                  ; MASK 00000080                                                         ;8
:203    ;       #100=28                 ; MASK 00000100                                                         ;8
:204    ;       #200=29                 ; MASK 00000200                                                         ;8
:205    ;       #400=2A                 ; MASK 00000400                                                         ;8
:206    ;       #800=2B                 ; MASK 00000800                                                         ;8
:207    ;       #1000=2C                ; MASK 00001000                                                         ;8
:208    ;       #2000=2D                ; MASK 00002000                                                         ;8
:209    ;       #4000=2E                ; MASK 00004000                                                         ;8
:210    ;       #8000=2F                ; MASK 00008000                                                         ;8
:211    ;       #10000=30               ; MASK 00010000                                                         ;8
:212    ;       #20000=31               ; MASK 00020000                                                         ;8
:213    ;       #40000=32               ; MASK 00040000                                                         ;8
:214    ;       #80000=33               ; MASK 00080000                                                         ;8
:215    ;       #100000=34              ; MASK 00100000                                                         ;8
:216    ;       #200000=35              ; MASK 00200000                                                         ;8
:217    ;       #400000=36              ; MASK 00400000                                                         ;8
:218    ;       #800000=37              ; MASK 00800000                                                         ;8
:219    ;       #1000000=38             ; MASK 01000000                                                         ;8
:220    ;       #2000000=39             ; MASK 02000000                                                         ;8
```

```
:221    ;       #4000000=3A             ; MASK 04000000
:222    ;       #8000000=3B             ; MASK 08000000
:223    ;       #10000000=3C            ; MASK 10000000
:224    ;       #20000000=3D            ; MASK 20000000
:225    ;       #40000000=3E            ; MASK 40000000
:226    ;       #80000000=3F            ; MASK 80000000
:227    ;
:228    ;       BIT0=20                 ; MASK 00000001
:229    ;       BIT1=21                 ; MASK 00000002
:230    ;       BIT2=22                 ; MASK 00000004
:231    ;       BIT3=23                 ; MASK 00000008
:232    ;       BIT4=24                 ; MASK 00000010
:233    ;       BIT5=25                 ; MASK 00000020
:234    ;       BIT6=26                 ; MASK 00000040
:235    ;       BIT7=27                 ; MASK 00000080
:236    ;       BIT8=28                 ; MASK 00000100
:237    ;       BIT9=29                 ; MASK 00000200
:238    ;       BIT10=2A                ; MASK 00000400
:239    ;       BIT11=2B                ; MASK 00000800
:240    ;       BIT12=2C                ; MASK 00001000
:241    ;       BIT13=2D                ; MASK 00002000
:242    ;       BIT14=2E                ; MASK 00004000
:243    ;       BIT15=2F                ; MASK 00008000
:244    ;       BIT16=30                ; MASK 00010000
:245    ;       BIT17=31                ; MASK 00020000
:246    ;       BIT18=32                ; MASK 00040000
:247    ;       BIT19=33                ; MASK 00080000
:248    ;       BIT20=34                ; MASK 00100000
:249    ;       BIT21=35                ; MASK 00200000
:250    ;       BIT22=36                ; MASK 00400000
:251    ;       BIT23=37                ; MASK 00800000
:252    ;       BIT24=38                ; MASK 01000000
:253    ;       BIT25=39                ; MASK 02000000
:254    ;       BIT26=3A                ; MASK 04000000
:255    ;       BIT27=3B                ; MASK 08000000
:256    ;       BIT28=3C                ; MASK 10000000
:257    ;       BIT29=3D                ; MASK 20000000
:258    ;       BIT30=3E                ; MASK 40000000
:259    ;       BIT31=3F                ; MASK 80000000
:260    ;
:261    ;       GPR0=40                 ; GPR 0
:262    ;       GPR1=41                 ; GPR 1
:263    ;       GPR2=42                 ; GPR 2
:264    ;       GPR3=43                 ; GPR 3
:265    ;       GPR4=44                 ; GPR 4
:266    ;       GPR5=45                 ; GPR 5
:267    ;       GPR6=46                 ; GPR 6
:268    ;       CM.SP=46                ; IN COMPATIBILITY THIS IS THE STACK POINTER.
:269    ;       GPR7=47                 ; GPR 7
:270    ;       CM.PC=47                ; IN COMPATIBILITY MODE THIS IS THE PC.
:271    ;       GPR8=48                 ; GPR 8
:272    ;       GPR9=49                 ; GPR 9
:273    ;       GPR10=4A                ; GPR 10
:274    ;       GPR11=4B                ; GPR 11
:275    ;       GPR12=4C                ; GPR 12
```

```
;276     ;       AP=4C                   ; ARGUMENT POINTER
;277     ;       GPR13=4D                ; GPR 13
;278     ;       FP=4D                   ; FRAME POINTER
;279     ;       SP=4E                   ; GPR 14
;280     ;       T0=4F                   ; TEMP LOCATION 0
;281     ;
;282     ;       ZERO=50                 ; ZERO CONSTANT
;283     ;       #3=51           ; CONSTANT
;284     ;       #5=52           ; CONSTANT
;285     ;       #6=53           ; CONSTANT
;286     ;       #7=54           ; CONSTANT
;287     ;       #9=55           ; CONSTANT
;288     ;       #B=56           ; CONSTANT
;289     ;       #C=57           ; CONSTANT
;290     ;       #D=58           ; CONSTANT
;291     ;       #E=59           ; CONSTANT
;292     ;       #F=5A           ; CONSTANT
;293     ;       #13=5B          ; CONSTANT
;294     ;       #1F=5C          ; CONSTANT
;295     ;       #34=5D          ; CONSTANT
;296     ;       #3B=5E          ; CONSTANT
;297     ;       #3F=5F          ; CONSTANT
;298     ;       #4B=60          ; CONSTANT
;299     ;       #66=61          ; CONSTANT
;300     ;       #A0=62          ; CONSTANT
;301     ;       #F0=63          ; CONSTANT
;302     ;       #FF=64          ; CONSTANT
;303     ;       #1FF=65         ; CONSTANT
;304     ;       #FFF=66         ; CONSTANT
;305     ;       #2001=67                ; CONSTANT
;306     ;       #3000=68                ; CONSTANT
;307     ;       #7F80=69                ; CONSTANT
;308     ;       #C000=6A                ; CONSTANT
;309     ;       #FFE0=6B                ; CONSTANT
;310     ;       #FFFF=6C                ; CONSTANT
;311     ;       #1F0000=6D              ; CONSTANT
;312     ;       #200001=6E              ; CONSTANT
;313     ;       #F27000=6F              ; CONSTANT
;314     ;       #3000000=70             ; CONSTANT
;315     ;       #41F0000=71             ; CONSTANT
;316     ;       #7000000=72             ; CONSTANT
;317     ;       #3FFFFE00=73            ; CONSTANT
;318     ;       #7F800000=74            ; CONSTANT
;319     ;       #7FFFFE00=75            ; CONSTANT
;320     ;       #7FFFFF0E=76            ; CONSTANT
;321     ;       #BF800001=77            ; CONSTANT
;322     ;
;323     ;       GPR.OS=78               ; HARDWARE INDEX TO GPRS
;324     ;       BIT.OS(3-0)=79          ; 16 LOCATION HARDWARE INDEX TO BIT MASKS
;325     ;       BIT.OS(4-0)=7B          ; 32 LOCATION HARDWARE INDEX TO BIT MASKS
;326     ;       OS=7C                   ; OS REGISTER
;327     ;       DEC.CON=7D              ; DECIMAL CARRIES
;328     ;       SIZE=7E                 ; SIZE REGISTER
;329     ;       MDT= 7E                 ; MEMORY REFERENCE DATA SIZE
;330     ;       INTERRUPT.VEC=7E        ; HARDWARE INTERRUPT VECTOR
```

```
;331     ;
;332     ;       THIS WORD IS THE HARDWARE CONDITION CODES.  THE CC'S CAN BE READ
;333     ;       OR ARE WRITTEN HERE.  NO OTHER BITS IN THE PSL ARE EFFECTED.
;334     ;
;335     ;       PSL.CC=7F               ; PSL CONDITION CODES
;336     ;
;337     ;XD.ADRS/=<16:9>,.VALIDITY=<XD.VAL>
;338     ;
;339     ;       SAVED.2ND.REQUEST=0     ; MM SAVED STATE FOR REQUEST.
;340     ;       T1=1                    ; TEMP LOCATION 1
;341     ;       T2=2                    ; TEMP LOCATION 2
;342     ;       T3=3                    ; TEMP LOCATION 3
;343     ;       T4=4                    ; TEMP LOCATION 4
;344     ;       T5=5                    ; TEMP LOCATION 5
;345     ;       T6=6                    ; TEMP LOCATION 6
;346     ;       T7=7                    ; TEMP LOCATION 7
;347     ;       T8=8                    ; TEMP LOCATION 8
;348     ;       T9=9                    ; TEMP LOCATION 9
;349     ;       T10=0A                  ; TEMP LOCATION 10
;350     ;       BACKUP.CM.PC=0A         ;COMPATIBILITY MODE KEEPS ORIGINAL PC HERE.
;351     ;       T11=0B                  ; TEMP LOCATION 11
;352     ;       T12=0C                  ; TEMP LOCATION 12
;353     ;       T13=0D                  ; TEMP LOCATION 13
;354     ;       T14=0E                  ; TEMP LOCATION 14
;355     ;       T15=0F                  ; TEMP LOCATION 15
;356     ;       PC=10                   ; MACRO INSTRUCTION PC
;357     ;       WR.BACKUP=11            ; BACKUP WORKING REGISTER FOR MOV MEM.DATA TO WR[].
;358     ;       SAVED.VAR=12            ; MM SAVED VAR LOCATION
;359     ;       SAVED.DATA=13           ; MM SAVED DATA LOCATION
;360     ;       SAVED.PTE.ADDRESS=14    ; MM SAVED PAGE TABLE ENTRY ADDRESS
;361     ;       SAVED.PTE=15            ; MM SAVED PAGE TABLE ENTRY
;362     ;       T16=16                  ; TEMP LOCATION 16
;363     ;       T17=17                  ; TEMP LOCATION 17
;364     ;       IE.TEMP4=18             ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION FOUR.
;365     ;       #FFFFF000=19                    ; CONSTANT
;366     ;       #FFFFFF00=1A                    ; CONSTANT
;367     ;       #FFFFFFFC=1B                    ; CONSTANT
;368     ;       #FFFFFFFF=1C                    ; CONSTANT
;369     ;
;370     ;       THIS LOCATION CONTAINS THE SOFT COPY OF THE PSL.  WHENEVER THE HARDWARE
;371     ;       VERSION OF THE PSL IS CHANGED (PSL.HW) THEN THIS CHANGE IS
;372     ;       REPORTED HERE.  ALL BITS OF THE PSL ARE REALLY LIKE THOSE HERE
;373     ;       EXCEPT FOR THE CONDITION CODES WHICH MUST BE OBTAINED
;374     ;       FROM PSL.CC.
;375     ;
;376     ;       PSL=1D                  ; SOFT COPY OF VAX PSL
;377     ;       VAR=1E                  ; VIRTUAL ADDRESS REGISTER
;378     ;       VAR2=1F                 ; VIRTUAL ADDRESS REGISTER
;379     ;
;380     ;
;381     ;       #1=20                   ; MASK 00000001 (HEX)
;382     ;       #2=21                   ; MASK 00000002
;383     ;       #4=22                   ; MASK 00000004
;384     ;       #8=23                   ; MASK 00000008
;385     ;       #10=24                  ; MASK 00000010
```

```
;386    ;       #20=25                  ; MASK 00000020
;387    ;       #40=26                  ; MASK 00000040
;388    ;       #80=27                  ; MASK 00000080
;389    ;       #100=28                 ; MASK 00000100
;390    ;       #200=29                 ; MASK 00000200
;391    ;       #400=2A                 ; MASK 00000400
;392    ;       #800=2B                 ; MASK 00000800
;393    ;       #1000=2C                ; MASK 00001000
;394    ;       #2000=2D                ; MASK 00002000
;395    ;       #4000=2E                ; MASK 00004000
;396    ;       #8000=2F                ; MASK 00008000
;397    ;       #10000=30               ; MASK 00010000
;398    ;       #20000=31               ; MASK 00020000
;399    ;       #40000=32               ; MASK 00040000
;400    ;       #80000=33               ; MASK 00080000
;401    ;       #100000=34              ; MASK 00100000
;402    ;       #200000=35              ; MASK 00200000
;403    ;       #400000=36              ; MASK 00400000
;404    ;       #800000=37              ; MASK 00800000
;405    ;       #1000000=38             ; MASK 01000000
;406    ;       #2000000=39             ; MASK 02000000
;407    ;       #4000000=3A             ; MASK 04000000
;408    ;       #8000000=3B             ; MASK 08000000
;409    ;       #10000000=3C            ; MASK 10000000
;410    ;       #20000000=3D            ; MASK 20000000
;411    ;       #40000000=3E            ; MASK 40000000
;412    ;       #80000000=3F            ; MASK 80000000
;413    ;
;414    ;       BIT0=20                 ; MASK 00000001
;415    ;       BIT1=21                 ; MASK 00000002
;416    ;       BIT2=22                 ; MASK 00000004
;417    ;       BIT3=23                 ; MASK 00000008
;418    ;       BIT4=24                 ; MASK 00000010
;419    ;       BIT5=25                 ; MASK 00000020
;420    ;       BIT6=26                 ; MASK 00000040
;421    ;       BIT7=27                 ; MASK 00000080
;422    ;       BIT8=28                 ; MASK 00000100
;423    ;       BIT9=29                 ; MASK 00000200
;424    ;       BIT10=2A                ; MASK 00000400
;425    ;       BIT11=2B                ; MASK 00000800
;426    ;       BIT12=2C                ; MASK 00001000
;427    ;       BIT13=2D                ; MASK 00002000
;428    ;       BIT14=2E                ; MASK 00004000
;429    ;       BIT15=2F                ; MASK 00008000
;430    ;       BIT16=30                ; MASK 00010000
;431    ;       BIT17=31                ; MASK 00020000
;432    ;       BIT18=32                ; MASK 00040000
;433    ;       BIT19=33                ; MASK 00080000
;434    ;       BIT20=34                ; MASK 00100000
;435    ;       BIT21=35                ; MASK 00200000
;436    ;       BIT22=36                ; MASK 00400000
;437    ;       BIT23=37                ; MASK 00800000
;438    ;       BIT24=38                ; MASK 01000000
;439    ;       BIT25=39                ; MASK 02000000
;440    ;       BIT26=3A                ; MASK 04000000
```

```
;441     ;       BIT27=3B                ; MASK 08000000
;442     ;       BIT28=3C                ; MASK 10000000
;443     ;       BIT29=3D                ; MASK 20000000
;444     ;       BIT30=3E                ; MASK 40000000
;445     ;       BIT31=3F                ; MASK 80000000
;446     ;
;447     ;       GPR0=40                 ; GPR 0
;448     ;       GPR1=41                 ; GPR 1
;449     ;       GPR2=42                 ; GPR 2
;450     ;       GPR3=43                 ; GPR 3
;451     ;       GPR4=44                 ; GPR 4
;452     ;       GPR5=45                 ; GPR 5
;453     ;       GPR6=46                 ; GPR 6
;454     ;       CM.SP=46                ; IN COMPATIBILITY MODE THIS IS THE STACK POINTER.
;455     ;       GPR7=47                 ; GPR 7
;456     ;       CM.PC=47                ; IN COMPATIBILITY MODE THIS IS THE PC.
;457     ;       GPR8=48                 ; GPR 8
;458     ;       GPR9=49                 ; GPR 9
;459     ;       GPR10=4A                ; GPR 10
;460     ;       GPR11=4B                ; GPR 11
;461     ;       GPR12=4C                ; GPR 12
;462     ;       AP=4C                   ; ARGUMENT POINTER
;463     ;       GPR13=4D                ; GPR 13
;464     ;       FP=4D                   ; FRAME POINTER
;465     ;       SP=4E                   ; GPR 14
;466     ;       T0=4F                   ; TEMP LOCATION 0
;467     ;
;468     ;       ZERO=50                 ; ZERO CONSTANT
;469     ;       #3=51           ; CONSTANT
;470     ;       #5=52           ; CONSTANT
;471     ;       #6=53           ; CONSTANT
;472     ;       #7=54           ; CONSTANT
;473     ;       #9=55           ; CONSTANT
;474     ;       #B=56           ; CONSTANT
;475     ;       #C=57           ; CONSTANT
;476     ;       #D=58           ; CONSTANT
;477     ;       #E=59           ; CONSTANT
;478     ;       #F=5A           ; CONSTANT
;479     ;       #13=5B          ; CONSTANT
;480     ;       #1F=5C          ; CONSTANT
;481     ;       #34=5D          ; CONSTANT
;482     ;       #3B=5E          ; CONSTANT
;483     ;       #3F=5F          ; CONSTANT
;484     ;       #4B=60          ; CONSTANT
;485     ;       #66=61          ; CONSTANT
;486     ;       #A0=62          ; CONSTANT
;487     ;       #F0=63          ; CONSTANT
;488     ;       #FF=64          ; CONSTANT
;489     ;       #1FF=65         ; CONSTANT
;490     ;       #FFF=66         ; CONSTANT
;491     ;       #2001=67                ; CONSTANT
;492     ;       #3000=68                ; CONSTANT
;493     ;       #7F80=69                ; CONSTANT
;494     ;       #C000=6A                ; CONSTANT
;495     ;       #FFE0=6B                ; CONSTANT
```

```
;496     ;       #FFFF=6C                ; CONSTANT
;497     ;       #1F0000=6D              ; CONSTANT
;498     ;       #200001=6E              ; CONSTANT
;499     ;       #F27000=6F              ; CONSTANT
;500     ;       #3000000=70             ; CONSTANT
;501     ;       #41F0000=71             ; CONSTANT
;502     ;       #7000000=72             ; CONSTANT
;503     ;       #3FFFFE00=73            ; CONSTANT
;504     ;       #7F800000=74            ; CONSTANT
;505     ;       #7FFFFE00=75            ; CONSTANT
;506     ;       #7FFFFF0E=76            ; CONSTANT
;507     ;       #BF800001=77            ; CONSTANT
;508     ;
;509     ;       GPR.OS=78               ; HARDWARE INDEX TO GPRS
;510     ;       BIT.OS(3-0)=79          ; 16 LOCATION HARDWARE INDEX TO BIT MASKS
;511     ;       BIT.OS(4-0)=7B          ; 32 LOCATION HARDWARE INDEX TO BIT MASKS
;512     ;       OS=7C                   ; OS REGISTER
;513     ;       DEC.CON=7D              ; DECIMAL CARRIES
;514     ;       SIZE=7E                 ; SIZE REGISTER
;515     ;       MDT= 7E                 ; MEMORY REFERENCE DATA SIZE
;516     ;       INTERRUPT.VEC=7E        ; HARDWARE INTERRUPT VECTOR
;517     ;
;518     ;       THIS LOCATION IS THE HARDWARE CONDITION CODES.  THE CC'S CAN BE
;519     ;       READ OR ARE WRITTEN HERE.  NO OTHER BITS IN THE PSL ARE EFFECTED.
;520     ;
;521     ;       PSL.CC=7F               ; PSL CONDITION CODES
;522
;523     ; THESE ADDRESSES ARE ONLY ACCESSIBLE WITH THE MOV MICRO INSTRUCTION.
;524
;525     ;       KSP=80                  ; KERNEL STACK POINTER
;526     ;       ESP=81                  ; EXECUTIVE STACK POINTER
;527     ;       SSP=82                  ; SUPERVISOR STACK POINTER
;528     ;       USP=83                  ; USER STACK POINTER
;529     ;       ISP=84                  ; INTERRUPT STACK POINTER
;530     ;       ASTLVL=85               ; AST LEVEL
;531     ;       PCBB=86                 ; PROCESS CONTROL BLOCK BASE
;532     ;       SCBB=87                 ; SYSTEM CONTROL BLOCK BASE
;533     ;       SISR=88                 ; SOFTWARE INTERRUPT SUMMARY REGISTER
;534     ;
;535     ;       CONSOLE.COMMAND=89      ; ONE BYTE COMMAND.
;536     ;       CONSOLE.MODIFIER=8A     ; ONE BYTE MODIFIER.
;537     ;       CONSOLE.ADDRESS=8B      ; FOUR BYTES OF ADDRESS.
;538     ;       CONSOLE.DATA=8C         ; FOUR BYTES OF DATA.
;539     ;       CONSOLE.STATUS=8D       ; ONE BYTE OF STATUS.
;540     ;       PSEUDO.NICR=8E          ; COPY OF NICR THAT 8085 HAS.
;541     ;       PSEUDO.ICR=8F           ; COPY OF ICR THAT 8085 HAS.
;542     ;       DEBUG.SAVE.WR0-90       ; TEMPORARY SAVE WR0.
;543     ;       DEBUG.SAVE.WR1=91       ; TEMPORARY SAVE WR1.
;544     ;       DEBUG.SAVE.ALU.CC=92    ; TEMPORARY SAVE CONDITION CODES.
;545     ;       MEMORY.SIZE=93          ; MEMORY SIZE OF THE MACHINE.
;546     ;       CRD.LOG=94              ; INFORMATION ABOUT LAST SOFT MEMORY ERROR.
;547     ;       PACKET.A=95             ; TOP OF PACKET.
;548     ;       PACKET.B=96             ; BOTTOM OF PACKET.
;549     ;       DEBUG.SAVE.ADDRESS=97   ; ADDRESS FOR EXAMINE/DEPOSIT.
;550     ;
```

```
;551     ;       SAVED.MDT=99            ; SAVED ADDRESS OF MDT FOR MMIE OPTIMIZATION.
;552     ;       P0BR=9A                 ; P0 BASE REGISTER
;553     ;       P0LR=9B                 ; P0 LENGTH REGISTER
;554     ;       P1BR=9C                 ; P1 BASE REGISTER
;555     ;       P1LR=9D                 ; P1 LENGTH REGISTER
;556     ;       SBR=9E                  ; SYSTEM BASE REGISTER
;557     ;       SLR=9F                  ; SYSTEM LENGTH REGISTER
;558     ;
;559     ;       PORT0=0A0       ; RESERVED FOR PORT.
;560     ;       PORT1=0A1       ; RESERVED FOR PORT.
;561     ;       PORT2=0A2       ; RESERVED FOR PORT.
;562     ;       PORT3=0A3       ; RESERVED FOR PORT.
;563     ;       PORT4=0A4       ; RESERVED FOR PORT.
;564     ;       PORT5=0A5       ; RESERVED FOR PORT.
;565     ;       PORT6=0A6       ; RESERVED FOR PORT.
;566     ;       PORT7=0A7       ; RESERVED FOR PORT.
;567     ;       PORT8=0A8       ; RESERVED FOR PORT.
;568     ;       PORT9=0A9       ; RESERVED FOR PORT.
;569     ;       PORT10=0AA      ; RESERVED FOR PORT.
;570     ;       PORT11=0AB      ; RESERVED FOR PORT.
;571     ;       PORT12=0AC      ; RESERVED FOR PORT.
;572     ;       PORT13=0AD      ; RESERVED FOR PORT.
;573     ;       PORT14=0AE      ; RESERVED FOR PORT.
;574     ;       PORT15=0AF      ; RESERVED FOR PORT.
;575     ;       PORT16=0B0      ; RESERVED FOR PORT.
;576     ;       PORT17=0B1      ; RESERVED FOR PORT.
;577     ;       PORT18=0B2      ; RESERVED FOR PORT.
;578     ;       PORT19=0B3      ; RESERVED FOR PORT.
;579     ;       PORT20=0B4      ; RESERVED FOR PORT.
;580     ;       PORT21=0B5      ; RESERVED FOR PORT.
;581     ;       PORT22=0B6      ; RESERVED FOR PORT.
;582     ;       PORT23=0B7      ; RESERVED FOR PORT.
;583     ;       PORT24=0B8      ; RESERVED FOR PORT.
;584     ;       PORT25=0B9      ; RESERVED FOR PORT.
;585     ;       PORT26=0BA      ; RESERVED FOR PORT.
;586     ;       PORT27=0BB      ; RESERVED FOR PORT.
;587     ;       PORT28=0BC      ; RESERVED FOR PORT.
;588     ;       PORT29=0BD      ; RESERVED FOR PORT.
;589     ;       PORT30=0BE      ; RESERVED FOR PORT.
;590     ;       PORT31=0BF      ; RESERVED FOR PORT.
;591     ;
;592     ;       XGPR0=0C0               ; BACKUP COPY OF GPR 0
;593     ;       XGPR1=0C1               ; BACKUP COPY OF GPR 1
;594     ;       XGPR2=0C2               ; BACKUP COPY OF GPR 2
;595     ;       XGPR3=0C3               ; BACKUP COPY OF GPR 3
;596     ;       XGPR4=0C4               ; BACKUP COPY OF GPR 4
;597     ;       XGPR5=0C5               ; BACKUP COPY OF GPR 5
;598     ;       XGPR6=0C6               ; BACKUP COPY OF GPR 6
;599     ;       XGPR7=0C7               ; BACKUP COPY OF GPR 7
;600     ;       XGPR8=0C8               ; BACKUP COPY OF GPR 8
;601     ;       XGPR9=0C9               ; BACKUP COPY OF GPR 9
;602     ;       XGPR10=0CA              ; BACKUP COPY OF GPR 10
;603     ;       XGPR11=0CB              ; BACKUP COPY OF GPR 11
;604     ;       XGPR12=0CC              ; BACKUP COPY OF GPR 12
;605     ;       XGPR13=0CD              ; BACKUP COPY OF GPR 13
```

```
:606    ;       XSP=0CE                 ; BACKUP COPY OF GPR 14
:607    ;
:608    ;       #14=0D0                 ; CONSTANT
:609    ;       #18=0D1                 ; CONSTANT
:610    ;       #1C=0D2                 ; CONSTANT
:611    ;       #24=0D3                 ; CONSTANT
:612    ;       #28=0D4                 ; CONSTANT
:613    ;       #2C=0D5                 ; CONSTANT
:614    ;       #2D=0D6                 ; CONSTANT
:615    ;       #30=0D7                 ; CONSTANT
:616    ;       #F8=0D8                 ; CONSTANT
:617    ;       #FC=0D9                 ; CONSTANT
:618    ;       #2D20=0DA                       ; CONSTANT
:619    ;       #140000=0DB                     ; CONSTANT
:620    ;       #150000=0DC                     ; CONSTANT
:621    ;       #160000=0DD                     ; CONSTANT
:622    ;       #170000=0DE                     ; CONSTANT
:623    ;       #180000=0DF                     ; CONSTANT
:624    ;       #1E0000=0E0                     ; CONSTANT
:625    ;       #40A10=0E1                      ; CONSTANT
:626    ;       #C0000E0=0E2                    ; CONSTANT
:627    ;       #0FFF0000=0E3                   ; CONSTANT
:628    ;       #B020FF00=0E4                   ; CONSTANT
:629    ;       #FFFFFF10=0E5                   ; CONSTANT
:630    ;       DEBUG.SAVE.T1=0E6       ; TEMPORARY SAVE T1.
:631    ;       DEBUG.SAVE.OS=0E7       ; TEMPORARY SAVE OS.
:632    ;       DEBUG.SAVE.SIZE=0E8     ; TEMPORARY SAVE SIZE VALUE.
:633    ;       SAVED.WR0=0E9           ; MM SAVED WR0
:634    ;       SAVED.WR1=0EA           ; MM SAVED WR1
:635    ;       SAVED.2ND.VAR=0EB       ; MM SAVED VAR 2
:636    ;       SAVED.ALU.CC=0EC                ; MM SAVED ALU CONDITION CODES
:637    ;       SAVED.SIZE=0ED          ; MM SAVED LS[SIZE]
:638    ;       SAVED.OS=0EE            ; MM SAVED LS[OS]
:639    ;       SAVED.REQUEST=0EF       ; MM SAVED REQUEST DATA
:640    ;       SAVED.CRD.LOG=0F0       ; MM SAVED LOCATION
:641    ;       OS.BAK=0F1              ; USED BY WARM FLOATING POINT.
:642    ;
:643    ;       THIS LOCATION IS FOR USE BY THE MFPR AND MTPR TO THE CONSOLE AND
:644    ;       TU58 CSR'S.  ALL OF THE INTERRUPT ENABLE BITS ARE HEREIN.
:645    ;
:646    ;       CONSOLE/PROGRAM I/O MODE FLAG - BIT<31> 0 - CONSOLE 1 - PROGRAM I/O MODE
:647
:648    ;       CRD RETRY IN PROGRESS - BIT<5>  0 - NOTHING DOING 1 - IN PROGRESS
:649    ;       MACHINE CHECK IN PROGRESS - BIT<4>
:650    ;       TU58 TRANSMIT CSR IE - BIT<3>
:651    ;       TU58 RECEIVER CSR IE - BIT<2>
:652    ;       CONSOLE TRANSMIT CSR IE - BIT<1>
:653    ;       CONSOLE TRANSMIT CSR IE - BIT<0>
:654    ;
:655    ;       CONSOLE.CSR.IES=0F2     ; CONSOLE CSR IE'S AND CONSOLE/PROGRAM I/O MODE FLAG.
:656    ;       IE.TEMP1=0F3            ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION ONE
:657    ;       IE.TEMP2=0F4            ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION TWO
:658    ;       IE.TEMP3=0F5            ; INTERRUPTS AND EXCEPTIONS TEMPORARY LOCATION THREE
:659    ;
:660    ;       XGPR.OS=0F8             ; HARDWARE INDEX TO XGPRS
```

```
;661     ;       PORT(3-0)=0F9           ; 16 LOCATION HARDWARE INDEX TO PORT REGISTERS.
;662     ;       PORT(4-0)=0FB           ; 32 LOCATION HARDWARE INDEX TO PORT REGISTERS.
;663     ;       CCR=0FC                 ; CONSOLE READ REGISTER
;664     ;       TIMER=0FD               ; INTERVAL TIMER VALUE.
;665     ;       ALU.CC=0FE              ; ALU CONDITION CODES
;666     ;
;667     ;       THIS LOCATION IS A WRITE ONLY REGISTER.  THE FOLLOWING BITS IN THE
;668     ;       PSL ARE AFFECTED:
;669     ;
;670     ;       COMPATIBILITY MODE - BIT<31>
;671     ;       CURRENT MODE - BITS<25:24>
;672     ;       INTERRUPT PRIORITY LEVEL (IPL) - BITS<20:16>
;673     ;       TRACE TRAP ENABLE (REALLY T OR TP) - BIT<4>
;674     ;       NEGATIVE CONDITION CODE - BIT<3>
;675     ;       ZERO CONDITION CODE - BIT<2>
;676     ;       OVERFLOW CONDITION CODE - BIT<1>
;677     ;       CARRY CONDITION CODE - BIT<0>
;678     ;
;679     ;       PSL.HW=0FF              ; HARDWARE PSL BITS
;680
;681
;682     ; DATA PATH CONTROL
;683     ;
;684     ; THIS FIELD IS BUILT FROM AN ADDRESS IN THE CCODE MEMORY.  THE CCODE MEMORY
;685     ; CONTAINS DATA PATH MICRO INSTRUCTIONS TO PRODUCE THE DESIRED OPERATION
;686     ; FOR THE UCODE MACROS.
;687
;688     ; DATA PATH CONTROL FIELD FOR THE 'BASIC' MICRO INSTRUCTION
;689     ;
;690     DP/=<21:16>,.VALIDITY=<DP.VAL>,.DEFAULT=<.SHIFT[.AND[<SDP/Q.CMP.LS>,0FF],-2]>
;691     ;
;692     ; THIS FIELD IS ONLY USED WITH THE INSTRUCTIONS WHICH CAN HAVE
;693     ; XCHG ATTACHED.
;694     ;
;695     DP.SMALL/=<20:16>,.VALIDITY=<DP.VAL>
;696     ;
;697     ; EXCHANGE ORIGINAL WR TO LS
;698     ;
;699     XCHG.BIT/=<21>,.DEFAULT=<0>
;700
;701             YES=1
;702             NO=0
;703
;704     ; DATA PATH CONTROL FIELD FOR  THE 'EXTENDED' MICRO INSTRUCTION
;705     ;
;706     XDP/=<19:14>,.VALIDITY=<A.VAL>
;707
;708     ; DATA PATH CONTROL FOR THE 'MOVE' MICRO INSTRUCTION
;709     ;
;710     MDP/=<19:17>,.VALIDITY=<XD.VAL>
;711
```

```
;712     .PAGE
;713     ; CONDITION CODE FIELD / DATA PATH CONTEXT
;714     ;
;715     CC/=<6:5>,.VALIDITY=<CC.VAL>,.DEFAULT=<CC/DT(LONG)&HOLD.ALU.CC>
;716
;717             DT(LONG)&HOLD.ALU.CC=0          ; USE LONG CONTEXT(32 BITS) AND HOLD ALU CONDITION CODES
;718             DT(LONG)&SET.ALU.CC=1           ; USE LONG CONTEXT(32 BITS) AND  SET ALU CONDITION CODES
;719             DT(SIZE)&SET.ALU.CC=2           ; USE CONTEXT IN THE SIZE REGISTER AND SET ALU CONDITION CODES
;720             DT(SIZE)&MAP.ALU.CC.TO.PSL=3    ; TRANSFER ALU CONDITION CODES TO PSL CONDITION CODES HARDWARE DEPENDENT
;721
;722
;723     ; MEMORY DATA TYPE FIELD
;724     ;
;725     MDT/=<6:5>,.VALIDITY=<DT.VAL>
;726
;727             BYTE=0                  ; SIZE = BYTE
;728             WORD=1                  ; SIZE = WORD
;729             SIZE=2                  ; SIZE = CONTENTS OF SIZE REGISTER
;730             LONG=3                  ; SIZE = LONG
;731
;732
;733     ; SHIFT INPUT FIELD
;734     ;
;735     ; THIS CONTROL FIELD DETERMINES THE SHIFT INPUTS FOR THE FOLLOWING OPERATIONS.
;736     ; THE DIRECTION OF THE SHIFT IS DETERMINED BY THE 2901 DESTINATION CODE.
;737     ;
;738     SI/=<13:11>,.VALIDITY=<A.VAL>
;739
;740             ROT.WR=0                ; ROTATE WR
;741             ROT.WR.Q=1              ; ROTATE WR AND Q
;742             ASH.WR=2                ; ARITHMETIC SHIFT OF WR
;743             ASH.WR.Q=3              ; ARITHMETIC SHIFT OF WR AND Q
;744             MUL.SHF=4               ; SIGNED MULTIPLY SHIFT OPERATION
;745             UMUL.SHF=5              ; UNSIGNED MULTIPLY SHIFT OPERATION
;746             SHF.WR.Q=6              ; LOGICAL SHIFT WR AND Q
;747
;748
;749     ; SKIP MICRO CONTROL FIELD
;750     ;
;751     ; THIS FIELD IS VALID FOR ALL INSTRUCTIONS EXCEPT
;752     ;
;753     ;               JUMP   MICRO INSTRUCTION
;754     ;               DECODE MICRO INSTRUCTION
;755     ;
;756     SCTL/=<4:0>,.VALIDITY=<.AND[<SCTL.VAL>,<SKIP.PAGE.VAL>]>,.DEFAULT=<SCTL/NO.SKIP>
;757
;758             N.CLR=0                 ; ALU N-BIT EQUAL ZERO
;759             NEQ=1                   ; ALU Z-BIT EQUAL ZERO
;760             BIT.SET=1               ; ALU Z-BIT EQUAL ZERO
;761             V.CLR=2                 ; ALU V-BIT EQUAL ZERO
;762             C.SET=3                 ; ALU C-BIT EQUAL ONE
;763             GEQU=3                  ; ALU C-BIT EQUAL ONE
;764             NOT.PSL.C=4             ; PSL C EQUAL ZERO
;765             BR.FALSE=5              ; BRANCH INSTRUCTION FALSE
;766             R.DST=6                 ; REGISTER MODE FLAG
```

```
;767             NO.INTERRUPT=7          ; NO INTERRUPT PENDING
;768             N.SET=8                 ; ALU N-BIT EQUAL ONE
;769             EQL=9                   ; ALU Z-BIT EQUAL ONE
;770             BITS.CLR=9              ; ALU Z-BIT EQUAL ONE
;771             V.SET=0A                ; ALU V-BIT EQUAL ONE
;772             C.CLR=0B                ; ALU C-BIT EQUAL ZERO
;773             LSSU=0B                 ; ALU C-BIT EQUAL ZERO
;774             STATE.ZERO.SET=0C       ; STATE REGISTER BIT ZERO TRUE
;775             STATE.ONE.SET=0D        ; STATE REGISTER BIT ONE TRUE
;776             STATE.ZERO.CLR=0E       ; STATE REGISTER BIT ZERO FALSE
;777             STATE.ONE.CLR=0F        ; STATE REGISTER BIT ONE FALSE
;778             RET.NOERR=10            ; RETURN+1 IF NO MEMORY ERROR
;779             JMP.Z.CLR=11            ; JUMP TO (STACK) IF ALU Z = 0
;780             POP.USTACK=12           ; DISCARD TOP STACK ENTRY
;781             JMP.C.SET=13            ; JUMP TO (STACK) IF ALU C=1
;782             RETURN=14               ; RETURN TO (USTACK)
;783             NO.SKIP=15              ; EXECUTE NEXT INSTRUCTION
;784             RETURN+1=16             ; RETURN TO (USTACK)+1
;785             LOOP.IF.NO.I.AND.SYNC=17; JUMP TO (STACK) IF NO INTERRUPT AND NO ACCELERATOR SYNC.
;786             CONSOLE.ATTN=18         ; CONSOLE ATTENTION
;787             CONSOLE.ACK=19          ; CONSOLE ACKNOWLEDGE
;788             NO.FAST.INTERRUPT=1A    ; NO FAST INTERRUPTS PENDING
;789             BACKUP.MASK.VALID=1B    ; REGISTER BACKUP MASK VALID.
;790             LSS=1C                  ; SIGNED LESS THAN.
;791             MEM.REF.OK=1D           ; NO MEMORY ERROR TRUE
;792             SKIP=1E                 ; EXECUTE ADDRESS PLUS ONE
;793             SYNC=1F                 ; ACCELERATOR SYNCHRONIZATION
;794
;795
;796     ; JUMP MICRO CONTROL FIELD
;797     ;
;798     ; THIS FIELD IS VALID FOR THE FOLLOWING INSTRUCTIONS
;799     ;
;800     ;               JUMP   MICRO INSTRUCTION
;801     ;               DECODE MICRO INSTRUCTION
;802     ;
;803     JCTL/=<3:0>,.VALIDITY=<.AND[<JCTL.VAL>,<JUMP.PAGE.VAL>]>,.DEFAULT=<JCTL/JUMP.VALID>
;804
;805             N.CLR=0                 ; ALU N-BIT EQUAL ZERO
;806             NEQ=1                   ; ALU Z-BIT EQUAL ZERO
;807             BIT.SET=1               ; ALU Z-BIT EQUAL ZERO
;808             V.CLR=2                 ; ALU V-BIT EQUAL ZERO
;809             C.SET=3                 ; ALU C-BIT EQUAL ONE
;810             GEQU=3                  ; ALU C-BIT EQUAL ONE
;811             JUMP=4                  ; JUMP UNCONDITIONALLY
;812             BR.FALSE=5              ; BRANCH INSTRUCTION FALSE
;813             R.DST=6                 ; REGISTER MODE FLAG
;814             NO.INTERRUPT=7          ; NO INTERRUPT PENDING
;815             N.SET=8                 ; ALU N-BIT EQUAL ONE
;816             EQL=9                   ; ALU Z-BIT EQUAL ONE
;817             BITS.CLR=9              ; ALU Z-BIT EQUAL ONE
;818             V.SET=0A                ; ALU V-BIT EQUAL ONE
;819             C.CLR=0B                ; ALU C-BIT EQUAL ZERO
;820             LSSU=0B                 ; ALU C-BIT EQUAL ZERO
;821             JSR=0C                  ; UNCONDITIONAL JUMP AND PUSH USTACK
```

```
;822             JSR.VALID=0D            ; JUMP AND PUSH USTACK IF IB VALID=1
;823             NO.JUMP.TST=0E          ; DO NOT JUMP AND SKIP IF IB VALID=0
;824             JUMP.VALID=0F           ; JUMP IF IB VALID=1
;825
;826
;827     ; JUMP ADDRESS FIELD
;828     ;
;829     JUMP.ADRS/=<17:4>,.ADDRESS,.VALIDITY=<JUMP.VAL>
;830
;831
;832     ; OS ENABLE
;833     ;
;834     OS/=<18>,.VALIDITY=<JUMP.VAL>
;835
;836             NOP=0
;837             WITH.OS=1               ; CONCATENATE OS REGISTER TO JUMP ADDRESS
;838
;839
```

```
;840     .PAGE
;841     ; BACKUP PC
;842     ;
;843     BPC/=<18>,.VALIDITY=<DECODE.VAL>,.DEFAULT=<.CASE[<.EQL[<OPC2/>,<OPC2/DECODE>]>]OF[0,1]>
;844
;845             NOP=1
;846             SAVE.PC=0               ; WRITE ALU DATA INTO WR[B]   (PC+1)
;847
;848
;849     ; DECODE ADDRESS FIELD
;850     ;
;851     DEC.ADRS/=<17:12>,.VALIDITY=<DECODE.VAL>
;852
;853
;854     ; IR FUNCTION
;855     ;
;856     IFUNC/=<11:9>,.VALIDITY=<DECODE.VAL>
;857
;858             CM.EXEC=0               ; COMPATIBILITY MODE EXECUTION DISPATCH WHEN UPPER BYTE = ZERO
;859             SPEC=0                  ; SPECIFIER DISPATCH
;860             CM.IRD=1                ; COMPATIBILITY MODE INSTRUCTION DECODE DISPATCH
;861             FLOAT=1                 ; SPECIFIER FLOAT TYPE DISPATCH
;862             VAX.EXEC=2              ; VAX EXECUTION DISPATCH
;863             ASRC=2                  ; ADDRESS SPECIFIER DISPATCH
;864             VAX.IRD=3               ; VAX OPCODE DISPATCH
;865             VSRC=4                  ; ADDRESS SPECIFIER DISPATCH FOR BIT FIELD INSTRUCTIONS
;866             ESRC=5                  ; SPECIFIER DISPATCH IN INDEX MODE
;867             CM.DST=6                ; COMPATIBILITY MODE DESTINATION DISPATCH
;868             CM.SINGLE=7             ; COMPATIBILITY MODE SOP DISPATCH
;869
;870
;871     ; INSTRUCTION BUFFER REQUEST
;872     ;
;873     IB.REQ/=<8>,.VALIDITY=<DECODE.VAL>,.DEFAULT=0
;874
;875             NOP=0
;876             IB.REQ=1                ; ENABLE HARDWARE DEPENDENT MEMORY REQUEST
;877     ;
;878     ; COMPATIBILITY MODE OPCODE SELECT
;879     ;
;880     CM.HI/=<7>,.VALIDITY=<DECODE.VAL>,.DEFAULT=0
;881
;882             LOW.BYTE=0              ; DECODE IR BITS<7:0>
;883             HI.BYTE=1               ; DECODE IR BITS <15:6>
;884
;885
;886     ; LOAD OS REGISTER
;887     ;
;888     LD.OS/=<6>,.VALIDITY=<DECODE.VAL>,.DEFAULT=0
;889
;890             NOP=0
;891             LOAD.OS=1               ; LOAD OS REGISTER FROM I-BUFFER
;892
;893
;894     ; REGISTER DESTINATION FLAG ENABLE
```

```
;895     ;
;896     R.DST/=<5>,.VALIDITY=<DECODE.VAL>,.DEFAULT=0
;897
;898             NOP=0
;899             ENAB=1
;900     ;
;901     ; OPCODE SPECIFIER BIT
;902     ;
;903     OPC.SPEC/=<4>,.VALIDITY=<DECODE.VAL>
;904
;905             CM.EXEC=1               ; COMPATIBILITY MODE EXECUTION DISPATCH
;906             SPEC=0                  ; SPECIFIER DISPATCH
;907             CM.IRD=1                ; COMPATIBILITY MODE INSTRUCTION DECODE DISPATCH
;908             FLOAT=0                 ; SPECIFIER FLOAT TYPE DISPATCH
;909             VAX.EXEC=1              ; VAX EXECUTION DISPATCH
;910             ASRC=0                  ; ADDRESS SPECIFIER DISPATCH
;911             VAX.IRD=1               ; VAX OPCODE DISPATCH
;912             VSRC=0                  ; ADDRESS SPECIFIER DISPATCH FOR BIT FIELD INSTRUCTIONS
;913             ESRC=0                  ; SPECIFIER DISPATCH IN INDEX MODE
;914             CM.DST=0                ; COMPATIBILITY MODE DESTINATION DISPATCH
;915             CM.SINGLE=0             ; COMPATIBILITY MODE SOP DISPATCH
```

```
;916     .PAGE
;917     ; MISCELLANEOUS FUNCTIONS
;918     ;
;919     ;
;920     ; THIS FIELD IS A FLAG FOR STEERING THE MICROWORD AS A MISCELLANEOUS
;921     ; INSTRUCTION OR AS A PORT INSTRUCTION.
;922     ;
;923     MISC.PORT/=<18>
;924
;925             MISC=0
;926             PORT=1
;927
;928     MISC.0/=<17>,.VALIDITY=<MISC.VAL>
;929
;930             NOP=0                       ; NO OPERATION IN ALU OR CONDITION CODES.
;931
;932     MISC.1/=<15:12>,.VALIDITY=<MISC.VAL>
;933
;934             NOP=0F                      ; NO OPERATION
;935             SET.CP.ATTN=0E              ; SET CPU ATTENTION FOR CONSOLE
;936             SET.CP.ACK=0D               ; SET CPU ACKNOWLEDGE FOR CONSOLE
;937             CLR.CP.ATTN.AND.ACK=0C      ; CLEAR CPU ATTENTION AND CPU ACKNOWLEDGE
;938             SET.HIGH.PAGE=0B            ; SET BIT FOR HIGH HALF OF WCS.
;939             CLR.HIGH.PAGE=0A            ; CLEAR BIT FOR LOW HALF OF WCS.
;940             SET.PORT.I.O=9              ; SET PORT I/O MODE
;941             SET.ACC.I.O=8               ; SET ACCELERATOR I/O MODE
;942             SET.BACKUP.MASK.VALID=7     ; SET REGISTER BACKUP MASK FLAG
;943             SET.S1=6                    ; SET STATE ONE BIT
;944             SET.S0=5                    ; SET STATE ZERO BIT
;945             CLR.S1=4                    ; CLEAR STATE ONE BIT
;946             CLR.S0=3                    ; CLEAR STATE ZERO BIT
;947             SET.S1&CLR.S0=2             ; SET STATE ONE BIT AND CLEAR STATE ZERO BIT
;948             CLR.S1&SET.S0=1             ; CLEAR STATE ONE AND SET STATE ZERO
;949             CLR=0                       ; CLEAR STATE ONE AND STATE ZERO BITS.
;950
;951
;952     MISC.2/=<11:9>,.VALIDITY=<MISC.VAL>
;953
;954             NOP=0                       ; NO OPERATION
;955             MASK.INTERRUPTS=1           ; MASK ALL INTERRUPTS (FOR DECODE NEXT CYCLE).
;956             ASSERT.DA.AND.PASS.WR=2;    ASSERT DATA AVAILABLE AND PASS WR[0] TO THE ACCELERATOR OR PORT.
;957             TRAP.ACC=3                  ; TRAP ACCELERATOR
;958             READ.ACC.UPC=4              ; READ ACCELERATOR PROGRAM COUNTER
;959             TRANSFER.GRANT=5            ; RECOGNIZE PORT REQUEST.
;960             MASK.HALT.AND.T.BIT=6       ; MASK HALT AND T-BIT TRAPS (FOR DECODE TWO CYCLES LATER)
;961             WRITE.WR.TO.CONSOLE.REGISTER=7  ; SEND BITS <7:0> TO 8085.
;962
;963     MISC.WR/=<8:7>
;964
;965     MISC.WRL/=<8>
;966     MISC.WRR/=<7>
;967
;968     PORT.SELECT/=<17:15>
;969
;970             IDC=7
```

```
;971
;972     PORT.FUNC/=<14:13>
;973
;974             READ=0
;975             WRITE=1
;976             CONTROL=2
;977
;978     R.W.FUNC/=<12:10>
;979
;980             CSR=0
;981             DAR=1
;982             DATA.BYTE=2
;983             DATA.WORD=3
;984             PATTERN=4
;985             POSITION=5
;986
;987     CNTL.FUNC/=<12:10>
;988
;989             CLEAR.FIFO.CNTR=0
;990             RESET.BR=1
;991             CLEAR.IDC=3
;992             SET.AUTO=4
;993             CLEAR.AUTO=5
;994             SEL.FIFO.A=6
;995             SEL.FIFO.B=7
;996
```

```
;997     .PAGE
;998     ; MEMORY REQUEST FIELD
;999     ;
;1000    ; THIS FIELD IS USED TO INDICATE THE DESIRED OPERATION TO THE MEMORY CONTROL.
;1001    ; NOTE THAT THIS IS A DUMMY FIELD. IT IS ACTUALLY MADE UP OF TWO FIELDS.
;1002    ; M.FUNC1 AND M.FUNC2
;1003    ;
;1004    ; THE FOLLOWING SPECIAL CONSIDERATIONS MUST BE MADE:
;1005    ;
;1006    ; ROTATE.BYTE.RIGHT - PERFORMS 9-BIT ROTATE AND THEREFORE THE
;1007    ; ANSWER MUST BE CLEANED UP WITH A ROL WR[].
;1008    ;
;1009    ; WORD.SWAP - PERFORMS A 15-BIT ROTATE AND THERE FORE
;1010    ; THE ANSWER MUST BE CLEANED UP WITH A ROL WR[].
;1011    ;
;1012    ; OCTA.WRITE.P - MUST BE LONGWORD ALLIGNED.
;1013    ;
;1014    ; OCTA.WR.V.WCHK - MUST BE LONGWORD ALLIGNED.  THIS DATA CAN
;1015    ; CROSS PAGE BOUNDARIES
;1016    ;
;1017    MEM.FUNC/=<7>,.VALIDITY=0
;1018
;1019    ;       PREFETCH=0              ; USED TO TAKE VAR AND ADD ONE BEFORE REFERENCE.
;1020            WRITE.TB=01             ; WRITE TRANSLATION BUFFER
;1021            TEST.V.RCHK=2           ; TEST WITH VIRTUAL ADDRESS AND READ ACCESS CHECK
;1022            READ.P=3                ; READ WITH PHYSICAL ADDRESS
;1023            WRITE.P=4               ; WRITE WITH PHYSICAL ADDRESS
;1024            TEST.V.WCHK=5           ; WRITE WITH VIRTUAL ADDRESS AND WRITE ACCESS CHECK
;1025            ROTATE.BYTE.RIGHT=6     ; ROTATE ADDRESS DATA 9 BITS RIGHT AND RETURN
;1026            WORD.SWAP=7             ; ROTATE ADDRESS DATA 15 BITS LEFT AND RETURN
;1027            READ.V.NOCHK=8          ; READ WITH VIRTUAL ADDRESS AND READ ACCESS CHECK
;1028            READ.V.WCHK=9           ; READ WITH VIRTUAL ADDRESS AND WRITE ACCESS CHECK
;1029            READ.V.RCHK=0A          ; READ WITH VIRTUAL ADDRESS AND READ ACCESS CHECK
;1030            READ.MAINT.VAR.INC=0B   ; TEST VAR INCREMENTING.
;1031            WRITE.V.NOCHK=0C        ; WRITE WITH VIRTUAL ADDRESS AND NO ACCESS CHECK
;1032            WRITE.V.WCHK=0D         ; WRITE WITH VIRTUAL ADDRESS AND WRITE ACCESS CHECK
;1033            IB.FILL=0E              ; READ WITH VIRTUAL ADDRESS AND READ ACCESS CHECK AND FILL IB
;1034            READ.V.RCHK.IFILL=0E    ; READ WITH VIRTUAL ADDRESS AND READ ACCESS CHECK AND FILL IB
;1035            WRITE.UBS.MAP=0F        ; WRITE TO THE UNIBUS MAP.
;1036
;1037            OCTA.WRITE.P=10         ; WRITE OCTA DATA WITH PHYSICAL ADDRESS.
;1038            WRITE.TB.STEP=11        ; WRITE TO TWO TB ENTRIES(WITH CORRECT ADDRESS)
;1039            READ.UBS.MAP=12         ; READ UNIBUS MAP
;1040            READ.TB=13              ; READ TRANSLATION BUFFER
;1041            READ.MAINT.ADRS=14      ; RPUTS VA ON UNIBUS LINES AND READS THIS DATA
;1042            MAINT.ECC.DATA=15       ; TEST ALL ECC FUNCTIONS.
;1043            READ.CSR=16             ; READ CONTROL REGISTER
;1044            READ.CNTRL.REG=16       ; READ CONTROL REGISTER
;1045            WRITE.CNTRL.REG=17      ; WRITE CONTROL REGISTER
;1046            WRITE.CSR=17            ; WRITE CONTROL REGISTER
;1047            OCTA.READ.P=18          ; READ OCTA DATA WITH PHYSICAL ADDRESS.
;1048            READ.V.WCHK.LOCKED=19   ; READ VIRTUAL ADDRESS AND WRITE ACCESS CHECK AND LOCKED
;1049            OCTA.READ.V.RCHK=1A     ; READ OCTA WITH VIRTUAL ADDRESS AND READ ACCESS CHECK.
;1050            READ.MAINT.BR.CHECK=1B  ; TESTS BRANCHING FUNCTION.
;1051            ISSUE.BG=1C             ; ISSUES BUS GRANT (4+ADDRESS<1:0>)
```

```
;1052            OCTA.WR.V.WCHK=1D       ; WRITE OCTA WITH VIRTUAL ADDRESS AND WRITE ACCESS CHECK.                    ;1
;1053            QUAD.READ.V.RCHK=1E     ; READ QUAD WITH VIRTUAL ADDRESS AND READ ACCESS CHECK.                      ;1
;1054            READ.MAINT.UBS=1F       ; PUTS VA ON UNIBUS LINES AND READS THIS AS DATA.                            ;1
;1055
;1056
;1057    M.FUNC2/=<18:16>,.VALIDITY=<DT.VAL>
;1058
;1059    M.FUNC1/=<8:7>,.VALIDITY=<DT.VAL>
;1060
;1061    PAR/=<23>,.DEFAULT=<.PARITY[<OPC2/>,<OS/>,<JUMP.ADRS/>,<JCTL/>]>
;1062    .UCODE
```

```
;1063    .PAGE   'MACRO DEFINITIONS                                      VER 00.02''
;1064
;1065    ;       THE FORMAT FOR THIS GROUP IS TWO ADDRESS WITH A MODIFY DESTINATION.
;1066    ;       THE FIRST OPERAND IS COMBINED WITH THE SECOND OPERAND AND THE
;1067    ;       RESULT WRITTEN TO THE SECOND OPERAND.  CMP AND BIT INSTRUCTIONS ARE
;1068    ;       READ ONLY.
;1069    ;
;1070    ;       DURING ALL ARITHMETIC AND LOGICAL INSTRUCTIONS THE ALU CC'S
;1071    ;       CAN BE LOADED BY ADDING THE
;1072    ;
;1073    ;               DT(SIZE)&SET.ALU.CC
;1074    ;
;1075    ;       MACRO TO THE MICROWORD.  THIS SPECIFIES THAT ALU CC'S ARE LOADED
;1076    ;       WITH THE 2901 CONDITIONS.  NO EXTERNAL MUNGING IS DONE.  IF THE
;1077    ;       DATA OPERATION IS WITH BYTES THEN THE CC'S ARE SET
;1078    ;       ACCORDING TO BITS 7-0. WORD USES BITS 15-0 AND LONGWORD USES BITS
;1079    ;       31-0.  THE FOLLOWING IS A DESCRIPTION OF THE ALU CC'S VALUE
;1080    ;       FOR EACH FUNCTION:
;1081    ;
;1082    ;       ADD -- BINARY ADDITION OF THE TWO OPERANDS
;1083    ;
;1084    ;       N - SIGN BIT
;1085    ;       Z - SET IF ANSWER IS ZERO.
;1086    ;       V - ARITHMETIC OVERFLOW
;1087    ;       C - CARRY
;1088    ;
;1089    ;       SUB -- BINARY ADDITION OF FIRST OPERAND AND TWO'S COMPLEMENT OF THE SECOND OPERAND.
;1090    ;
;1091    ;       N - SIGN BIT
;1092    ;       Z - SET IF ANSWER IS ZERO.
;1093    ;       V - ARITHMETIC OVERFLOW
;1094    ;       C - COMPLEMENT OF THE BORROW.
;1095    ;
;1096    ;       BIS -- LOGICAL OR OF THE TWO OPERANDS.
;1097    ;
;1098    ;       N - SIGN BIT
;1099    ;       Z - SET IF ANSWER IS ZERO
;1100    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1101    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1102    ;
;1103    ;       BIC -- ONE'S COMPLEMENT OF FIRST OPERAND ANDED WITH SECOND OPERAND.
;1104    ;
;1105    ;       N - SIGN BIT
;1106    ;       Z - SET IF ANSWER IS ZERO
;1107    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1108    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1109    ;
;1110    ;       AND -- LOGICAL AND OF THE TWO OPERANDS.
;1111    ;
;1112    ;       N - SIGN BIT
;1113    ;       Z - SET IF ANSWER IS ZERO
;1114    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1115    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1116    ;
;1117    ;       XOR -- LOGICAL EXCLUSIVE OR OF THE TWO OPERANDS.
```

```
;1118   ;
;1119   ;       N - SIGN BIT
;1120   ;       Z - SET IF ANSWER IS ZERO
;1121   ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1122   ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1123   ;
;1124   ;       CMP -- PERFORM BINARY ADDITION OF FIRST OPERAND AND TWO'S COMPLEMENT OF SECOND OPERAND BUT DO NOT STORE.
;1125   ;
;1126   ;       N - SIGN BIT
;1127   ;       Z - SET IF ANSWER IS ZERO.
;1128   ;       V - ARITHMETIC OVERFLOW
;1129   ;       C - CARRY
;1130   ;
;1131   ;       BIT -- PERFORM LOGICAL AND OF THE TWO OPERANDS BUT DO NOT STORE.
;1132   ;
;1133   ;       N - SIGN BIT
;1134   ;       Z - SET IF ANSWER IS ZERO
;1135   ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1136   ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1137   ;
;1138   ;       SWAP -- SWITCH THE TWO VALUES.
;1139   ;
;1140   ;       N - SIGN BIT OF VALUE TO WR.
;1141   ;       Z - SET IF ANSWER IS ZERO OF VALUE TO WR.
;1142   ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1143   ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
```

```
;1144    .PAGE
;1145    ;
;1146    ;       MACROS FOR THE FORM OF WR[B]<-- WR[B] OPERATE LS[D]
;1147    ;
;1148    ADD  LS[] TO   WR[]              "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP.SMALL/<.SHIFT[<.AND[<SDP/LS.PLUS.WR>,07F]>,-2]>"
;1149    SUB  LS[] FROM WR[]              "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP.SMALL/<.SHIFT[<.AND[<SDP/LS.FROM.WR>,07F]>,-2]>"
;1150    BIS  LS[] TO   WR[]              "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/LS.OR.WR>,0FF]>,-2]>"
;1151    BIC  LS[] TO   WR[]              "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/LS.MASK.WR>,0FF]>,-2]>"
;1152    AND  LS[] TO   WR[]              "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP.SMALL/<.SHIFT[<.AND[<SDP/LS.AND.WR>,07F]>,-2]>"
;1153    XOR  LS[] TO   WR[]              "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP.SMALL/<.SHIFT[<.AND[<SDP/LS.XOR.WR>,07F]>,-2]>"
;1154
;1155    CMP  LS[] WITH WR[]              "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/LS.CMP.WR>,0FF]>,-2]>"
;1156    BIT  LS[] WITH WR[]              "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.BIT.LS>,0FF]>,-2]>"
;1157    SWAP LS[] WITH WR[]              "OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/SWAP.LS.WR>,0FF]>,-2]>"
;1158    ;
;1159    ;       THE FOLLOWING MACROS IS TO BE USED WITH ADD,SUB,
;1160    ;       AND,XOR LS[] TO WR[]
;1161    ;
;1162    ;       IT WILL TAKE THE ORIGINAL CONTENTS OF WR[]
;1163    ;       AND PUT IT IN LS[].
;1164    ;
;1165    XCHG                             "XCHG.BIT/YES"
;1166    ;
;1167    ;       MACROS FOR THE FORM OF LS[D]<-- LS[D] OPERATE Q-REGISTER
;1168    ;
;1169    ADD    Q TO   LS[]               "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/Q.PLUS.LS>,0FF]>,-2]>"
;1170    SUB    Q FROM LS[]               "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/Q.FROM.LS>,0FF]>,-2]>"
;1171    BIS    Q TO   LS[]               "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/Q.OR.LS>,0FF]>,-2]>"
;1172    AND    Q TO   LS[]               "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/Q.AND.LS>,0FF]>,-2]>"
;1173    XOR    Q TO   LS[]               "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/Q.XOR.LS>,0FF]>,-2]>"
;1174
;1175    CMP    Q WITH LS[]               "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/Q.CMP.LS>,0FF]>,-2]>"
;1176    BIT    Q WITH LS[]               "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/LS.BIT.Q>,0FF]>,-2]>"
;1177    ;
;1178    ;       MACROS FOR THE FORM OF Q-REGISTER<-- Q-REGISTER OPERATE LS[D]
;1179    ;
;1180    ADD  LS[] TO   Q                 "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/LS.PLUS.Q>,0FF]>,-2]>"
;1181    SUB  LS[] FROM Q                 "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/LS.FROM.Q>,0FF]>,-2]>"
;1182    BIS  LS[] TO   Q                 "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/LS.OR.Q>,0FF]>,-2]>"
;1183    BIC  LS[] TO   Q                 "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/LS.MASK.Q>,0FF]>,-2]>"
;1184    AND  LS[] TO   Q                 "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/LS.AND.Q>,0FF]>,-2]>"
;1185    XOR  LS[] TO   Q                 "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/LS.XOR.Q>,0FF]>,-2]>"
;1186
;1187    CMP  LS[] WITH Q                 "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/LS.CMP.Q>,0FF]>,-2]>"
;1188    BIT  LS[] WITH Q                 "OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/LS.BIT.Q>,0FF]>,-2]>"
;1189    ;
;1190    ;       MACROS FOR THE FORM OF LS[D]<-- LS[D] OPERATE WR[B]
;1191    ;
;1192    ADD  WR[] TO   LS[]              "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.PLUS.LS>,0FF]>,-2]>"
;1193    SUB  WR[] FROM LS[]              "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.FROM.LS>,0FF]>,-2]>"
;1194    BIS  WR[] TO   LS[]              "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.OR.LS>,0FF]>,-2]>"
;1195    AND  WR[] TO   LS[]              "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.AND.LS>,0FF]>,-2]>"
;1196    XOR  WR[] TO   LS[]              "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.XOR.LS>,0FF]>,-2]>"
;1197
;1198    CMP  WR[] WITH LS[]              "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.CMP.LS>,0FF]>,-2]>"
```

```
;1199   BIT  WR[] WITH LS[]          "OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.BIT.LS>,0FF]>,-2]>"
;1200   SWAP WR[] WITH LS[]          "SWAP LS[a2] WITH WR[a1]"
;1201   ;
;1202   ;       MACROS FOR THE FORM OF WR[B]<-- WR[B] OPERATE WR[A]
;1203   ;
;1204   ADD  WR[] TO   WR[]          "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.PLUS.WR>,03F]>"
;1205   SUB  WR[] FROM WR[]          "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.FROM.WR>,03F]>"
;1206   BIS  WR[] TO   WR[]          "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.OR.WR>,03F]>"
;1207   BIC  WR[] TO   WR[]          "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.MASK.WR>,03F]>"
;1208   AND  WR[] TO   WR[]          "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.AND.WR>,03F]>"
;1209   XOR  WR[] TO   WR[]          "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.XOR.WR>,03F]>"
;1210
;1211   CMP  WR[] WITH WR[]          "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.CMP.WR>,03F]>"
;1212   BIT  WR[] WITH WR[]          "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.BIT.WR>,03F]>"
;1213   ;
;1214   ;       MACROS FOR THE FORM OF WR[B]<-- WR[B] OPERATE Q-REGISTER
;1215   ;
;1216   ADD  Q TO   WR[]             "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a1,XDP/<.AND[<SDP/Q.PLUS.WR>,03F]>"
;1217   SUB  Q FROM WR[]             "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a1,XDP/<.AND[<SDP/Q.FROM.WR>,03F]>"
;1218   BIS  Q TO   WR[]             "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a1,XDP/<.AND[<SDP/Q.OR.WR>,03F]>"
;1219   AND  Q TO   WR[]             "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a1,XDP/<.AND[<SDP/Q.AND.WR>,03F]>"
;1220   XOR  Q TO   WR[]             "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a1,XDP/<.AND[<SDP/Q.XOR.WR>,03F]>"
;1221
;1222   CMP  Q WITH WR[]             "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a1,XDP/<.AND[<SDP/Q.CMP.WR>,03F]>"
;1223   BIT  Q WITH WR[]             "OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a1,XDP/<.AND[<SDP/Q.BIT.WR>,03F]>"
;1224   ;
;1225   ;       MACROS FOR THE FORM OF Q-REGISTER<-- Q-REGISTER OPERATE WR[B]
;1226   ;
;1227   ADD  WR[] TO   Q             "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/WR.PLUS.Q>,03F]>"
;1228   SUB  WR[] FROM Q             "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/WR.FROM.Q>,03F]>"
;1229   BIS  WR[] TO   Q             "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/WR.OR.Q>,03F]>"
;1230   BIC  WR[] TO   Q             "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/WR.MASK.Q>,03F]>"
;1231   AND  WR[] TO   Q             "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/WR.AND.Q>,03F]>"
;1232   XOR  WR[] TO   Q             "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/WR.XOR.Q>,03F]>"
;1233
;1234   CMP  WR[] WITH Q             "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/WR.CMP.Q>,03F]>"
;1235   BIT  WR[] WITH Q             "OPC1/EXTENDED,A.ADRS/a1,XDP/<.AND[<SDP/Q.BIT.WR>,03F]>"
```

```
;1236    .PAGE
;1237    ;
;1238    ;       THE FORMAT OF THIS GROUP IS THREE ADDRESS TYPE. THE FIRST AND SECOND
;1239    ;       OPERANDS ARE READ AND THE RESULT IS WRITTEN INTO THE THIRD OPERAND.
;1240    ;
;1241    ;       MACROS FOR THE FORM OF Q-REGISTER<-- WR[B] OPERATE WR[A]
;1242    ;
;1243    ADD  WR[] PLUS WR[] TO Q        "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.PLUS.WR.Q>,03F]>"
;1244    SUB  WR[] FROM WR[] TO Q        "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.FROM.WR.Q>,03F]>"
;1245    BIS  WR[] WITH WR[] TO Q        "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.OR.WR.Q>,03F]>"
;1246    BIC  WR[] WITH WR[] TO Q        "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.MASK.WR.Q>,03F]>"
;1247    AND  WR[] WITH WR[] TO Q        "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.AND.WR.Q>,03F]>"
;1248    XOR  WR[] WITH WR[] TO Q        "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.XOR.WR.Q>,03F]>"
;1249    ;
;1250    ;       MACROS FOR THE FORM OF Q-REGISTER<-- WR[B] OPERATE LS[D]
;1251    ;
;1252    ADD  WR[] PLUS LS[] TO Q        "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.PLUS.LS.Q>,0FF]>,-2]>"
;1253    SUB  WR[] FROM LS[] TO Q        "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.FROM.LS.Q>,0FF]>,-2]>"
;1254    BIS  WR[] WITH LS[] TO Q        "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.OR.LS.Q>,0FF]>,-2]>"
;1255    AND  WR[] WITH LS[] TO Q        "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.AND.LS.Q>,0FF]>,-2]>"
;1256    XOR  WR[] WITH LS[] TO Q        "OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.XOR.LS.Q>,0FF]>,-2]>"
;1257    ;
;1258    ;       MACROS FOR THE FORM OF Q-REGISTER <-- LS[D] OPERATE WR[B]
;1259    ;
;1260    ADD LS[] PLUS WR[] TO Q "ADD WR[@2] PLUS LS[@1] TO Q"
;1261    SUB  LS[] FROM WR[] TO Q        "OPC/BASIC,B.ADRS/@2,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/LS.FROM.WR.Q>,0FF]>,-2]>"
;1262    BIS LS[] WITH WR[] TO Q "BIS WR[@2] WITH LS[@1] TO Q"
;1263    AND LS[] WITH WR[] TO Q         "AND WR[@2] WITH LS[@1] TO Q"
;1264    XOR LS[] WITH WR[] TO Q         "XOR WR[@2] WITH LS[@1] TO Q"
;1265
;1266
```

;

```
;1267    .PAGE
;1268    ; SPECIAL FUNCTION ADD/SUB OPERATIONS
;1269
;1270    ADD (WR[] TO WR[])+1            ''OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.PLUS.WR+1>,03F]>''
;1271    ADD (LS[] TO WR[])+1            ''OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/LS.PLUS.WR+1>,0FF]>,-2]>''
;1272    ADD (WR[] TO LS[])+1            ''OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.PLUS.LS+1>,0FF]>,-2]>''
;1273
;1274    ADD OS PLUS WR[] TO LS[]        ''OPC1/MOVE,B.ADRS/a1,XD.ADRS/a2,MDP/<.SHIFT[<.AND[<SDP/WR.PLUS.OS>,3F]>,-3]>''
;1275
;1276    SUB (WR[] FROM WR[])-1          ''OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.FROM.WR-1>,03F]>''
;1277    SUB (LS[] FROM WR[])-1          ''OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/LS.FROM.WR-1>,0FF]>,-2]>''
;1278    SUB (WR[] FROM LS[])-1          ''OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WR.FROM.LS-1>,0FF]>,-2]>''
;1279
;1280
;1281    SUB  WRB[] FROM WRA[] TO WRB[]  ''OPC1/EXTENDED,B.ADRS/a1,A.ADRS/a2,XDP/<.AND[<SDP/WR.FROM.WR.WR>,03F]>''
;1282    SUB  LS[]  FROM WR[]  TO LS     ''OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/LS.FROM.WR.LS>,0FF]>,-2]>''
;1283    SUB  WR[] FROM LS[]  TO WR      ''OPC/BASIC,B.ADRS/a1,D.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/WRA.FROM.LS.WRB>,0FF]>,-2]>''
;1284
;1285    SUB  WRA[] FROM Q    TO WRB[]   ''OPC1/EXTENDED,A.ADRS/a1,B.ADRS/a2,XDP/<.AND[<SDP/WR.FROM.Q.WR>,03F]>''
;1286    SUB  LS[]  FROM Q    TO LS[]    ''OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/LS.FROM.Q.LS>,0FF]>,-2]>''
;1287    SUB  Q     FROM WR[] TO Q       ''OPC1/EXTENDED,B.ADRS/a1,XDP/<.AND[<SDP/Q.FROM.WR.Q>,03F]>''
;1288    SUB  Q     FROM LS[] TO Q       ''OPC/BASIC,D.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/Q.FROM.LS.Q>,0FF]>,-2]>''
;1289
;1290    ADD Q PLUS WR[] TO LS[]         ''OPC/BASIC,D.ADRS/a2,B.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/Q.PLUS.WR.TO.LS>,0FF]>,-2]>''
;1291    SUB Q FROM WR[] TO LS[]         ''OPC/BASIC,D.ADRS/a2,B.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/Q.FROM.WR.TO.LS>,0FF]>,-2]>''
;1292    ADD Q PLUS LS[] TO WR[] ''OPC/BASIC,D.ADRS/a1,B.ADRS/a2,DP/<.SHIFT[<.AND[<SDP/Q.PLUS.LS.TO.WR>,0FF]>,-2]>''
;1293
;1294    ADD Q TO WR[] XCHG TO LS[]      ''OPC/BASIC,D.ADRS/a2,B.ADRS/a1,DP/<.SHIFT[<.AND[<SDP/WR.PLUS.Q.XCHG>,0FF]>,-2]>''
;1295
```

```
;1296    .PAGE
;1297    ; MOVE MACRO INSTRUCTIONS
;1298    ;
;1299    ;       DURING THE MOVE INSTRUCTIONS THE ALU CC'S CAN BE LOADED BY
;1300    ;       ADDING THE
;1301    ;
;1302    ;               DT(SIZE)&SET.ALU.CC
;1303    ;
;1304    ;       MACRO TO THE MICROWORD.  THE FOLLOWING IS A DESCRIPTION OF THE ALU
;1305    ;       CC'S VALUE FOR EACH FUNCTION:
;1306    ;
;1307    ;       MOV -- PUT FIRST OPERAND ON TOP OF THE SECOND OPERAND.
;1308    ;
;1309    ;       N - SIGN BIT
;1310    ;       Z - SET IF ANSWER IS ZERO
;1311    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1312    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1313    ;
;1314    ;       MCOM -- PUT ONE'S COMPLEMENT OF THE FIRST OPERAND ON TOP OF THE SECOND OPERAND.
;1315    ;
;1316    ;       N - SIGN BIT
;1317    ;       Z - SET IF ANSWER IS ZERO
;1318    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1319    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1320    ;
;1321    ;       MNEG -- PUT TWO'S COMPLEMENT OF THE FIRST OPERAND ON TOP OF THE SECOND OPERAND.
;1322    ;
;1323    ;       N - SIGN BIT
;1324    ;       Z - SET IF ANSWER IS ZERO.
;1325    ;       V - ARITHMETIC OVERFLOW
;1326    ;       C - CARRY
;1327    MOV Q     TO LS[]               ''OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/MOV.Q.LS>,0FF]>,-2]>''
;1328    MOV Q     TO WR[]               ''OPC1/EXTENDED,B.ADRS/@1,XDP/<.AND[<SDP/MOV.Q.WR>,03F]>''
;1329    MOV LS[] TO Q                   ''OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/MOV.LS.Q>,0FF]>,-2]>''
;1330    MOV Q TO WR[] XCHG TO LS[]      ''OPC/BASIC,D.ADRS/@2,B.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/MOV.Q.WR&WR.LS>,0FF]>,-2]>''
;1331
;1332    MOV IB.DATA TO OS               ''OPC2/DECODE,IFUNC/SPEC,R.DST/NOP,IB.REQ/IB.REQ,JCTL/NO.JUMP.TST,LD.OS/LOAD.OS''
;1333    MOV CM.IB.DATA TO OS            ''OPC2/DECODE,IFUNC/SPEC,R.DST/NOP,IB.REQ/NOP,JCTL/NO.JUMP.TST,LD.OS/LOAD.OS''
;1334
;1335    MOV MEM.DATA TO WR[]            ''OPC1/MOVE,B.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.MEM.WR>,3F]>,-3]>,XD.ADRS/WR.BACKUP''
;1336    MOV MEM.DATA TO WR[] XCHG TO LS[]       ''OPC1/MOVE,B.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.MEM.WR>,3F]>,-3]>,XD.ADRS/@2''
;1337    MOV MEM.DATA TO LS[]            ''OPC1/MOVE,XD.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.MEM.LS>,3F]>,-3]>''
;1338    MOV LS[]     TO WR[]            ''OPC1/MOVE,XD.ADRS/@1,B.ADRS/@2,MDP/<.SHIFT[<.AND[<SDP/MOV.LS.WR>,3F]>,-3]>''
;1339    MOV WR[]     TO LS[]            ''OPC1/MOVE,B.ADRS/@1,XD.ADRS/@2,MDP/<.SHIFT[<.AND[<SDP/MOV.WR.LS>,3F]>,-3]>''
;1340    MOV WR[]     TO WR[]            ''OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.PLUS.0.TO.WR>,03F]>''
;1341    MOV WR[]     TO Q               ''OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.OR.WR.Q>,03F]>''
;1342    MOV XWR[]    TO  Q              ''OPC1/MOVE,XB.ADRS/@1,XD.ADRS/0,MDP/<.SHIFT[<.AND[<SDP/MOV.XWR.Q>,3F]>,-3]>''
;1343
;1344    MOV FPA TO LS[]                 ''OPC1/MOVE,XD.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.ACC.LS>,3F]>,-3]>''
;1345    MOV PORT TO LS[]                ''OPC1/MOVE,XD.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.ACC.LS>,3F]>,-3]>,CC/DT(LONG)&HOLD.ALU.CC''
;1346
;1347    MCOM WR[] TO LS[]               ''OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.COM.LS>,0FF]>,-2]>''
;1348    MCOM LS[] TO WR[]               ''OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/LS.COM.WR>,0FF]>,-2]>''
;1349    MNEG WR[] TO LS[]               ''OPC/BASIC,B.ADRS/@1,D.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/WR.NEG.LS>,0FF]>,-2]>''
;1350    MNEG LS[] TO WR[]               ''OPC/BASIC,D.ADRS/@1,B.ADRS/@2,DP/<.SHIFT[<.AND[<SDP/LS.NEG.WR>,0FF]>,-2]>''
```

```
;1351   MNEG WR[] TO WR[]              ''OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.NEG.WR>,03F]>''
;1352   MCOM WR[] TO WR[]       ''OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.COM.WR>,03F]>''
;1353   MINC WR[] TO WR[]       ''OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.INC.WR>,03F]>''
;1354   MDEC WR[] TO WR[]       ''OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,XDP/<.AND[<SDP/WR.DEC.WR>,03F]>''
;1355   MNEG Q TO LS[]          ''OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/Q.NEG.LS>,0FF]>,-2]>''
```

```
;1356    .PAGE
;1357    ;
;1358    ; SINGLE OPERAND OPERATIONS
;1359    ;
;1360    ;       THIS FORM OF INSTRUCTION PERFORMS THE DESIRED OPERATION ON THE OPERAND
;1361    ;       SPECIFIED.
;1362    ;
;1363    ;               CLEAR
;1364    ;               NEGATE
;1365    ;               INCREMENT
;1366    ;               DECREMENT
;1367    ;               COMPLEMENT
;1368    ;               TEST
;1369    ;
;1370    ;       DURING THE SINGLE OPERAND INSTRUCTIONS THE ALU CC'S CAN BE
;1371    ;       LOADING USING THE
;1372    ;
;1373    ;               DT(SIZE)&SET.ALU.CC
;1374    ;
;1375    ;       MACRO IN THE MICROWORD.  THE FOLLOWING IS A DESCRIPTION OF THE
;1376    ;       ALU CC'S FOR EACH FUNCTION:
;1377    ;
;1378    ;       CLR -- STORE A ZERO IN THE OPERAND.
;1379    ;
;1380    ;       N - SIGN BIT
;1381    ;       Z - SET IF ANSWER IS ZERO
;1382    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1383    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1384    ;
;1385    ;       NEG -- PERFORM TWO'S COMPLEMENT OF THE OPERAND.
;1386    ;
;1387    ;       N - SIGN BIT
;1388    ;       Z - SET IF ANSWER IS ZERO.
;1389    ;       V - ARITHMETIC OVERFLOW
;1390    ;       C - CARRY
;1391    ;
;1392    ;       INC -- ADD ONE TO THE OPERAND.
;1393    ;
;1394    ;       N - SIGN BIT
;1395    ;       Z - SET IF ANSWER IS ZERO.
;1396    ;       V - ARITHMETIC OVERFLOW
;1397    ;       C - CARRY
;1398    ;
;1399    ;       DEC -- SUBTRACT ONE FROM THE OPERAND.
;1400    ;
;1401    ;       N - SIGN BIT
;1402    ;       Z - SET IF ANSWER IS ZERO.
;1403    ;       V - ARITHMETIC OVERFLOW
;1404    ;       C - CARRY
;1405    ;
;1406    ;       COM -- PERFORM ONE'S COMPLEMENT OF THE OPERAND (XNOR WITH ZERO).
;1407    ;
;1408    ;       N - SIGN BIT
;1409    ;       Z - SET IF ANSWER IS ZERO.
;1410    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
```

```
;1411    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1412    ;
;1413    ;       TST -- ADD ZERO TO THE OPERAND TO SET THE CONDITION CODES.
;1414    ;
;1415    ;       N - SIGN BIT
;1416    ;       Z - SET IF ANSWER IS ZERO
;1417    ;       V - ARITHMETIC OVERFLOW (IN THIS CASE ALWAYS ZERO)
;1418    ;       C - CARRY (IN THIS CASE ZERO)
;1419    ;
;1420
;1421    CLR Q                           "OPC1/EXTENDED,A.ADRS/0,B.ADRS/0,XDP/<.AND[<SDP/WR.XOR.WR.Q>,03F]>"
;1422    CLR LS[]                        "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/CLR.LS>,0FF]>,-2]>"
;1423    CLR WR[]                        "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.XOR.WR>,03F]>"
;1424
;1425    NEG  Q                          "OPC1/EXTENDED,XDP/<.AND[<SDP/NEG.Q>,03F]>"
;1426    NEG  LS[]                       "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/NEG.LS>,0FF]>,-2]>"
;1427    NEG  WR[]                       "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.NEG.WR>,03F]>"
;1428
;1429    INC  Q                          "OPC1/EXTENDED,XDP/<.AND[<SDP/INC.Q>,03F]>"
;1430    INC  LS[]                       "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/INC.LS>,0FF]>,-2]>"
;1431    INC  WR[]                       "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.INC.WR>,03F]>"
;1432
;1433    DEC  Q                          "OPC1/EXTENDED,XDP/<.AND[<SDP/DEC.Q>,03F]>"
;1434    DEC  LS[]                       "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/DEC.LS>,0FF]>,-2]>"
;1435    DEC  WR[]                       "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.DEC.WR>,03F]>"
;1436
;1437    COM  Q                          "OPC1/EXTENDED,XDP/<.AND[<SDP/COM.Q>,03F]>"
;1438    COM  LS[]                       "OPC/BASIC,D.ADRS/@1,DP/<.SHIFT[<.AND[<SDP/COM.LS>,0FF]>,-2]>"
;1439    COM  WR[]                       "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.COM.WR>,03F]>"
;1440
;1441    TST WR[]                        "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,XDP/<.AND[<SDP/WR.PLUS.0.TO.WR>,03F]>"
;1442    TST Q                           "OPC1/EXTENDED,XDP/<.AND[<SDP/Q.PLUS.0>,03F]>"
;1443
;1444    NOP                             "OPC/BASIC,D.ADRS/0,B.ADRS/0,DP/<.SHIFT[<.AND[<SDP/Q.CMP.LS>,0FF]>,-2]>"
```

```
;1445    .PAGE
;1446    ;
;1447    ; MACROS FOR SHIFT OPERATION ON WR[B] AND/OR Q-REGISTER
;1448    ;
;1449    ;       DURING THE SHIFT OPERATIONS THE ALU CC'S CAN BE LOADED BY
;1450    ;       USING THE
;1451    ;
;1452    ;               DT(SIZE)&SET.ALU.CC
;1453    ;
;1454    ;       MACRO WITH THE MICROWORD.  THE FOLLOWING IS A DESCRIPTION OF THE
;1455    ;       ALU CC'S WITH EACH FUNCTION.
;1456    ;
;1457    ;       ROR WR[] -- ROTATE 32 BIT VALUE ONE POSITION RIGHT.
;1458    ;
;1459    ;       N - SIGN OF INPUT
;1460    ;       Z - SET IF INPUT IS ZERO
;1461    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1462    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1463    ;
;1464    ;       ROL WR[] -- ROTATE 32 BIT VALUE ONE POSITION LEFT.
;1465    ;
;1466    ;       N - SIGN OF INPUT
;1467    ;       Z - SET IF INPUT IS ZERO
;1468    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1469    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1470    ;
;1471    ;       ASHR WR[] -- SHIFT 32 BIT VALUE RIGHT ONE POSITION AND SIGN EXTEND.
;1472    ;
;1473    ;       N - SIGN OF INPUT
;1474    ;       Z - SET IF INPUT IS ZERO
;1475    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1476    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1477    ;
;1478    ;       ASHL WR[] -- SHIFT 32 BIT VALUE LEFT ONE POSITION AND INSERT A ZERO IN BOTTOM BIT.
;1479    ;
;1480    ;       N - SIGN OF INPUT
;1481    ;       Z - SET IF INPUT IS ZERO
;1482    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1483    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1484    ;
;1485    ;       ASHRC WR[].Q -- SHIFT 64 BIT WR.Q ONE POSITION LEFT AND SIGN EXTEND.
;1486    ;
;1487    ;       N - SIGN OF INPUT WR[]
;1488    ;       Z - SET IF INPUT WR[] IS ZERO
;1489    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1490    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1491    ;
;1492    ;       RORC WR[].Q -- ROTATE 64 BIT WR.Q ONE POSITION RIGHT.
;1493    ;
;1494    ;       N - SIGN OF INPUT WR[]
;1495    ;       Z - SET IF INPUT WR[] IS ZERO
;1496    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1497    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1498    ;
;1499    ;       ASHLC WR[].Q -- SHIFT 64 BIT WR.Q ONE POSITION LEFT AND INSERT ZERO.
```

```
;1500    ;
;1501    ;       N - SIGN OF INPUT WR[]
;1502    ;       Z - SET IF INPUT WR[] IS ZERO
;1503    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1504    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1505    ;
;1506    ;       ROLC WR[].Q -- ROTATE 64 BIT WR.Q ONE POSITION LEFT.
;1507    ;
;1508    ;       N - SIGN OF INPUT WR[]
;1509    ;       Z - SET IF INPUT WR[] IS ZERO
;1510    ;       V - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1511    ;       C - RANDOM GARBAGE (DO NOT ASSUME ANY FUNCTION)
;1512    ;
;1513    ;       SHL2 WR[] -- SHIFT 32 BIT VALUE TWO POSITIONS LEFT INSERTING ZEROES IN THE BOTTOM BITS.
;1514    ;
;1515    ;       N - SIGN OF INPUT WR[]+WR[]
;1516    ;       Z - SET IF INPUT WR[]+WR[] IS ZERO
;1517    ;       V - OVERFLOW OF INPUT WR[]+WR[]
;1518    ;       C - CARRY OF INPUT WR[]+WR[]
;1519    ;
;1520    ROR  WR[]                       "OPC1/EXTENDED,B.ADRS/@1,SI/ROT.WR,XDP/<.AND[<SDP/ASHR>,03F]>"
;1521
;1522    ROL  WR[]                       "OPC1/EXTENDED,B.ADRS/@1,SI/ROT.WR,XDP/<.AND[<SDP/ASHL>,03F]>"
;1523
;1524
;1525    ASHR  WR[]                      "OPC1/EXTENDED,B.ADRS/@1,SI/ASH.WR,XDP/<.AND[<SDP/ASHR>,03F]>"
;1526
;1527    ASHL  WR[]                      "OPC1/EXTENDED,B.ADRS/@1,SI/ASH.WR,XDP/<.AND[<SDP/ASHL>,03F]>"
;1528
;1529    ASHRC  WR[].Q                   "OPC1/EXTENDED,B.ADRS/@1,SI/ASH.WR.Q,XDP/<.AND[<SDP/ASHRC>,03F]>"
;1530
;1531    RORC WR[].Q                     "OPC1/EXTENDED,B.ADRS/@1,SI/ROT.WR.Q,XDP/<.AND[<SDP/ASHRC>,03F]>"
;1532
;1533    ASHLC  WR[].Q                   "OPC1/EXTENDED,B.ADRS/@1,SI/ASH.WR.Q,XDP/<.AND[<SDP/ASHLC>,03F]>"
;1534
;1535    ROLC WR[].Q                     "OPC1/EXTENDED,B.ADRS/@1,SI/ROT.WR.Q,XDP/<.AND[<SDP/ASHLC>,03F]>"
;1536
;1537    SHL2  WR[]                      "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@1,SI/2,XDP/<.AND[<SDP/SHL2>,03F]>"
;1538
;1539    COND.ADD&SHIFT  WR[] TO WR[].Q  "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,SI/MUL.SHF,XDP/<.AND[<SDP/COND.ADD&SHIFT>,03F]>"
;1540
;1541    COND.SUB&SHIFT  WR[] FROM WR[].Q "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,SI/MUL.SHF,XDP/<.AND[<SDP/COND.SUB&SHIFT>,03F]>"
;1542
;1543    UNSIGNED.MUL WR[] TO WR[].Q     "OPC1/EXTENDED,A.ADRS/@1,B.ADRS/@2,SI/UMUL.SHF,XDP/<.AND[<SDP/COND.ADD&SHIFT>,03F]>"
;1544    SHR WR[]                        "OPC1/EXTENDED,B.ADRS/@1,SI/SHF.WR.Q,XDP/<.AND[<SDP/ASHR>,03F]>"
;1545    SHL WR[]                        "ASHL WR[@1]"
;1546    SHRC WR[].Q                     "OPC1/EXTENDED,B.ADRS/@1,SI/SHF.WR.Q,XDP/<.AND[<SDP/ASHRC>,03F]>"
;1547    SHLC WR[].Q                     "ASHLC WR[].Q"
;1548
;1549
;1550    ; MEMORY REQUEST MACRO
;1551
;1552    ; *      THIS MACRO IS NOT FOR GENERAL USE!!!            *
;1553    ; *      THIS IS ONLY FOR USE IN THE FOLLOWING MACRO     *
;1554    MEM.FUNC[]                      "M.FUNC2/<.SHIFT[<MEM.FUNC/@1>,-2]>,M.FUNC1/<.AND[<MEM.FUNC/@1>,3]>"
```

```
;1555
;1556   MEM.REQ[] ADRS[] DT[]          ''OPC2/MEM.REQ,MEM.FUNC[@1],D.ADRS/@2,MDT/@3''
;1557   WRITE.MEM LS[]                 ''OPC1/MOVE,XD.ADRS/@1,MDP/<.SHIFT[<.AND[<SDP/MOV.LS.MEM>,3F]>,-3]>''
```

```
;1558    .PAGE
;1559    ;
;1560    ; MACROS FOR MISCELLANEOUS OPERATIONS
;1561    ;
;1562    ;       IN THE NEBULA MICROPROGRAMMING LANGUAGE THERE IS A NEED FOR
;1563    ;       VARIOUS UNIQUE FUNCTIONAL OPERATIONS.  THESE ARE FOUND IN THE
;1564    ;       MISC INSTRUCTIONS.  MOST FUNCTIONS ARE EXECUTED BY INSERTING
;1565    ;       THE REQUEST INSIDE THE MISC[] AND MISC2[] MACROS.
;1566    ;
;1567    MISC []                         ''OPC2/MISC,MISC.1/@1,MISC.2/NOP,MISC.PORT/MISC''
;1568    MISC2 []                        ''OPC2/MISC,MISC.1/NOP,MISC.2/@1,MISC.PORT/MISC''
;1569    MISC [] WR[]                    'MISC [@1],MISC.WRL/0,MISC.WRR/@2''
;1570    MISC2 [] WR[]                   'MISC2 [@1],MISC.WRL/0,MISC.WRR/@2''
;1571
;1572    PORT.CONTROL []                 ''OPC2/MISC,MISC.PORT/PORT,CNTL.FUNC/@1,PORT.SELECT/IDC,PORT.FUNC/CONTROL''
;1573    PORT.WRITE PORT.ADRS[] WITH WR[]        ''OPC2/MISC,MISC.PORT/PORT,R.W.FUNC/@1,PORT.SELECT/IDC,PORT.FUNC/WRITE,MISC.WRL/0,MIS
;1574    PORT.READ PORT.ADRS[]   ''OPC2/MISC,MISC.PORT/PORT,R.W.FUNC/@1,PORT.SELECT/IDC,PORT.FUNC/READ''
;1575    ;
;1576    ;       A FEW CASES OF THE MISCELLANEOUS FUNCTIONS HAVE BEEN CALLED OUT AS
;1577    ;       UNIQUE FUNCTIONS IN THE MICROPROGRAMMING LANGUAGE SET.  THESE
;1578    ;       ARE LISTED BELOW.
;1579    ;
;1580    ;       THE FOLLOWING TO MACROS SEND DATA TO EXTERNAL ACCELLERATORS.
;1581    ;
;1582    ASSERT.DA.AND.PASS.WR0          'MISC2 [ASSERT.DA.AND.PASS.WR] WR[0]''
;1583    ASSERT.DA.AND.PASS.WR1          'MISC2 [ASSERT.DA.AND.PASS.WR] WR[1]''
;1584    ;
;1585    ;       STATE REGISTER MACROS - THE POSSIBLE CHANGES TO THE STATE
;1586    ;       BITS ARE SPECIFIED AS INDIVIDUAL FUNCTIONS.
;1587    ;
;1588    CLR.STATE.ZERO                  ''OPC2/MISC,MISC.1/CLR.S0,MISC.2/NOP,MISC.PORT/MISC''
;1589    CLR.STATE.ONE                   ''OPC2/MISC,MISC.1/CLR.S1,MISC.2/NOP,MISC.PORT/MISC''
;1590    SET.STATE.ZERO                  ''OPC2/MISC,MISC.1/SET.S0,MISC.2/NOP,MISC.PORT/MISC''
;1591    SET.STATE.ONE                   ''OPC2/MISC,MISC.1/SET.S1,MISC.2/NOP,MISC.PORT/MISC''
;1592    SET.STATE.ZERO&CLR.STATE.ONE    ''OPC2/MISC,MISC.1/CLR.S1&SET.S0,MISC.2/NOP,MISC.PORT/MISC''
;1593    SET.STATE.ONE&CLR.STATE.ZERO    ''OPC2/MISC,MISC.1/SET.S1&CLR.S0,MISC.2/NOP,MISC.PORT/MISC''
;1594    CLR.STATE.ONE&CLR.STATE.ZERO    ''OPC2/MISC,MISC.1/CLR,MISC.2/NOP,MISC.PORT/MISC''
;1595    SET.BACKUP.MASK.VALID           ''OPC2/MISC,MISC.1/SET.BACKUP.MASK.VALID,MISC.2/NOP,MISC.PORT/MISC''
;1596
;1597
;1598    ; CONTEXT SIZE AND CONDITION CODE MACROS
;1599
;1600    DT(SIZE)&MAP.ALU.CC.TO.PSL      ''CC/DT(SIZE)&MAP.ALU.CC.TO.PSL''
;1601    DT(LONG)&SET.ALU.CC             ''CC/DT(LONG)&SET.ALU.CC''
;1602    DT(SIZE)&SET.ALU.CC             ''CC/DT(SIZE)&SET.ALU.CC''
;1603
```

```
;1604    .PAGE
;1605    ; JUMP AND MICRO SEQUENCER CONTROL MACROS
;1606    ;
;1607    ;       THE FOLLOWING MACROS ARE USED TO CONTROL THE PROGRAMS FLOW OF
;1608    ;       CONTROL.
;1609    ;
;1610    ;       JMP [] -- TRANSFER TO NEW ADDRESS UNCONDITIONALLY.
;1611    ;
;1612    ;       JMP.OS [] -- TRANSFER TO RESULTANT ADDRESS OF LOGICAL OR OF ADDRESS
;1613    ;               IN THE INSTRUCTION AND THE OS REGISTER UNCONDITIONALLY.
;1614    ;
;1615    ;       JMP.IF[] TO [] -- TRANSFER TO NEW ADDRESS ONLY IF JUMP
;1616    ;               CONDITION IS MET.
;1617    ;
;1618    ;       JSR [] -- TRANSFER TO NEW ADDRESS UNCONDITIONALLY AND PUT
;1619    ;               RETURN ADDRESS ON THE SEQUENCER STACK.
;1620    ;
;1621    ;       JSR.OS [] -- TRANSFER TO RESULTANT ADDRESS OF LOGICAL OR OF
;1622    ;               ADDRESS IN THE INSTRUCTION AND THE OS REGISTER.
;1623    ;               THE RETURN ADDRESS IS ALSO PUSHED ON THE SEQUENCER STACK.
;1624    ;
;1625    ;       RETURN -- TRANSFER TO ADDRESS ON TOP OF THE SEQUENCER STACK AND
;1626    ;                POP THE STACK.
;1627    ;
;1628    ;       RETURN+1 -- TRANSFER TO ADDRESS ON TOP OF THE SEQUENCER
;1629    ;               STACK PLUS ONE AND POP THE STACK.
;1630    ;
;1631    ;       RETURN+1.IF(MEM.REF.OK) -- IF THE MOST RECENT MEMORY REQUEST HAD
;1632    ;               NO ERROR THEN TRANSFER TO ADDRESS ON TOP OF THE SEQUENCER
;1633    ;               STACK PLUS ONE AND POP THE STACK.  IF THERE WAS AN
;1634    ;               ERROR THEN SKIP OVER THE ENEXT INSTRUCTION.
;1635    ;
;1636    ;       LOOP.IF(NEQ) -- IF THE ALU Z-BIT IS ZERO (LAST VALUE CALCULATED
;1637    ;               WHEN THE ALU.CC'S WERE SET DID NOT RESULT AS ZERO)
;1638    ;               THEN TRANSFER TO THE ADDRESS ON TOP OF THE SEQUENCER
;1639    ;               STACK.  THE STACK IS NOT POPPED.  IF THE ALU Z-BIT
;1640    ;               IS ONE THEN CONTINUE WITH THE NEXT INSTRUCTION.
;1641    ;               (UPC+1) AND THE STACK IS POPPED.
;1642    ;
;1643    ;       LOOP.IF(GEQU) -- IF THE ALU C-BIT IS ONE (LAST VALUE CALCULATED WHEN
;1644    ;               THE ALU.CC'S WERE SET WAS GREATER THAN OR EQUAL TO ZERO)
;1645    ;               THEN TRANSFER TO THE ADDRESS ON TOP OF THE SEQUENCER
;1646    ;               STACK.  THE STACK IS NOT POPPED.  IF THE ALU C-BIT
;1647    ;               IS ZERO THEN CONTINUE WITH THE NEXT INSTRUCTION
;1648    ;               (UPC+1) AND THE STACK IS POPPED.
;1649    ;
;1650    ;       LOOP.IF(NO INTERRUPT AND NO SYNC) -- IF THE INTERRUPT CONDITION
;1651    ;               IS NOT TRUE AND THE ACCELERATOR SYNC CONDITION IS NOT
;1652    ;               TRUE THEN TRANSFER TO THE ADDRESS ON THE TOP OF THE SEQUENCER
;1653    ;               STACK.  THE STACK IS NOT POPPED.  IF THE INTERRUPT
;1654    ;               CONDITION IS TRUE THEN CONTINUE WITH THE NEXT INSTRUCTION
;1655    ;               (UPC+1) AND THE SEQUENCER STACK IS NOT POPPED.  IF THE ACCELERATOR
;1656    ;               SYNC IS TRUE THEN CONTINUE WITH THE NEXT INSTRUCTION
;1657    ;               (UPC+1) AND THE SEQUENCER STACK IS POPPED.
;1658    ;
```

```
;1659   ;       SKIP.IF[] -- IF THE SKIP CONDITION IS SATISFIED THEN TRANSFER
;1660   ;               TO UPC+2 ELSE TRANSFER TO UPC+1.
;1661   ;
;1662
;1663   JMP []                          "OPC2/JUMP,JUMP.ADRS/@1,JCTL/JUMP"
;1664   JMP.OS []                       "OPC2/JUMP,JUMP.ADRS/@1,OS/WITH.OS,JCTL/JUMP"
;1665   JMP.IF[] TO []                  "OPC2/JUMP,JCTL/@1,JUMP.ADRS/@2"
;1666   JSR []                          "OPC2/JUMP,JCTL/JSR,JUMP.ADRS/@1"
;1667   JSR.OS []                       "OPC2/JUMP,JCTL/JSR,JUMP.ADRS/@1,OS/WITH.OS"
;1668
;1669   RETURN                          "SCTL/RETURN"
;1670   RETURN+1                        "SCTL/RETURN+1"
;1671   RETURN+1.IF(MEM.REF.OK)         "SCTL/RET.NOERR"
;1672
;1673   LOOP.IF(NEQ)                    "SCTL/JMP.Z.CLR"
;1674   LOOP.IF(GEQU)                   "SCTL/JMP.C.SET"
;1675   LOOP.IF(NO INTERRUPT AND NO SYNC) ELSE SKIP.IF(SYNC)    "SCTL/LOOP.IF.NO.I.AND.SYNC"
;1676
;1677   SKIP.IF[]                       "SCTL/@1"
;1678   SKIP                            "SCTL/SKIP"
```

```
;1679    .PAGE
;1680    ;
;1681    ; INSTRUCTION STREAM MACROS
;1682    ;
;1683    ; *      THIS MACRO IS NOT FOR GENERAL USE!!!              *
;1684    ; *      THIS IS ONLY FOR USE IN THE FOLLOWING MACROS      *
;1685    DEC[]                                   'OPC2/DECODE,DEC.ADRS/<.SHIFT[<JUMP.ADRS/@1>,-8]>''
;1686
;1687    DECODE.SPEC ADRS[]                      'DEC[@1],IFUNC/SPEC,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/S
;1688    DECODE.ASRC ADRS[]                      'DEC[@1],IFUNC/ASRC,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/A
;1689    DECODE.ESRC ADRS[]                      'DEC[@1],IFUNC/ESRC,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/E
;1690    DECODE.VSRC ADRS[]                      'DEC[@1],IFUNC/VSRC,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/V
;1691    DECODE.FLOAT ADRS[]                     'DEC[@1],IFUNC/FLOAT,BPC/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,LD.OS/LOAD.OS,R.DST/ENAB,OPC.SPEC/
;1692    DECODE.OPC1 ADRS[]                      'DEC[@1],IFUNC/VAX.IRD,R.DST/NOP,CM.HI/LOW.BYTE,IB.REQ/IB.REQ,OPC.SPEC/VAX.IRD''
;1693    EXEC.DISPATCH ADRS[]                    'DEC[@1],IFUNC/VAX.EXEC,IB.REQ/NOP,R.DST/NOP,CM.HI/LOW.BYTE,JCTL/JSR,OPC.SPEC/VAX.EXEC''
;1694
;1695    DECODE.CM.OPC ADRS[]                    'DEC[@1],IFUNC/CM.IRD,CM.HI/HI.BYTE,OPC.SPEC/CM.IRD,LD.OS/LOAD.OS''
;1696    DECODE.CM.DST ADRS[]                    'DEC[@1],IFUNC/CM.DST,OPC.SPEC/CM.DST,LD.OS/LOAD.OS,R.DST/ENAB''
;1697    DECODE.CM.SINGLE ADRS[]                 'DEC[@1],IFUNC/CM.SINGLE,OPC.SPEC/CM.SINGLE''
;1698    CM.EXEC ADRS[]                          'DEC[@1],IFUNC/CM.EXEC,JCTL/JUMP,OPC.SPEC/CM.EXEC,LD.OS/LOAD.OS''
;1699
;1700    SAVE.PC WR0                             'BPC/SAVE.PC''
;1701    SAVE.PC WR4                             'BPC/SAVE.PC''
;1702    SAVE.CM.PC WR0                          'BPC/SAVE.PC''
;1703    SAVE.CM.PC WR4                          'BPC/SAVE.PC''
;1704
;1705
;1706    ; THESE SKIP CONDITIONS ARE USED FOR THE DECODE MICRO INSTRUCTION
;1707    ;
;1708    SKIP.IF(IB VALID)                       ''JCTL/NO.JUMP.TST''
;1709    JMP.IF(IB VALID)                        ''JCTL/JUMP.VALID''
;1710    JSR.IF(IB VALID)                        ''JCTL/JSR.VALID''
;1711    ;
;1712    ;       THESE SKIP CONDITIONS ARE TO BE USED WITH THE COMPATIBILITY MODE DECODE
;1713    ;       MACROS.
;1714    ;
;1715    JMP.TO []                               ''JCTL/JUMP,DEC[@1]''
;1716    JSR.TO []                               ''JCTL/JSR,DEC[@1]''
            ;1717    .BIN
```

```
;1718   .PAGE
;1719   .TOC    "FIELD DEFINTITIONS FOR DATA PATH CONTROL MICRO WORD    VER 00.01"
;1720
;1721   ; THIS SECTION IS FOR THE DATA PATH CONTROL ROMS
;1722   ;
;1723
;1724   .CCODE
;1725   SDP/=<8:0>,.ADDRESS,.VALIDITY=0
;1726   ;
;1727   ; THIS FIELD CORRESPONDS TO THE 2901 ALU FUNCTION CONTROL PINS ... I5,I4,I3
;1728   ;
;1729   ALU/=<2:0>,.DEFAULT=<ALU/R.OR.S>
;1730
;1731           R.PLUS.S=0              ; ADD R WITH S
;1732           S.MINUS.R=1             ; SUBTRACT R FROM S
;1733           R.MINUS.S=2             ; SUBTRACT S FROM R
;1734           R.OR.S=3                ; OR R WITH S
;1735           R.AND.S=4               ; AND R WITH S
;1736           NOTR.AND.S=5            ; COMPLEMENT R THEN AND WITH S
;1737           R.XOR.S=6               ; XOR R WITH S
;1738           R.XNOR.S=7              ; XOR R WITH S THEN COMPLEMENT THE RESULT
;1739   SRC/=<8:6>,.DEFAULT=<SRC/D.0>
;1740
;1741           0.Q=2                   ; RMX=0         SMX=Q
;1742           0.B=3                   ; RMX=0         SMX=B
;1743           A.Q=0                   ; RMX=A         SMX=Q
;1744           A.B=01                  ; RMX=A         SMX=B
;1745           D.Q=06                  ; RMX=D         SMX=Q
;1746           D.0=7                   ; RMX=D         SMX=0
;1747           0.A=4                   ; RMX=0         SMX=A
;1748           D.A=5                   ; RMX=D         SMX=A
;1749   ;
;1750   ; THIS FIELD CORRESPONDES TO THE 2901 ALU DESTINATION
;1751   ; CONTROL PINS I8, I7 AND I6.
;1752   ;
;1753   DST/=<5:3>,.DEFAULT=<DST/NOP>
;1754
;1755           LOAD.Q=0                ; LOAD Q-REGISTER
;1756           NOP=1                   ; HOLD ALL REGISTERS
;1757           WRITE.B.A=2             ; WRITE RAM B-PORT AND OUTPUT RAM A-PORT
;1758           WRITE.B.F=3             ; WRITE RAM B-PORT AND OUTPUT ALU
;1759           RSHF.RAM.Q=4            ; RIGHT SHIFT RAM AND Q
;1760           RSHF.RAM=5              ; RIGHT SHIFT RAM
;1761           LSHF.RAM.Q=6            ; LEFT SHIFT RAM AND Q
;1762           LSHF.RAM=7              ; LEFT SHIFT RAM
;1763
;1764   CIN/=<9>,.DEFAULT=0
;1765
;1766           NO.CIN=0                ; NO CARRY IN TO ALU
;1767           CIN=1                   ; CARRY IN TO ALU
;1768
;1769
;1770
```

```
                ;1771   .PAGE
                ;1772   .TOC    "CONTROL ROM CONTENTS                                   VER 00.01"
                ;1773   .SEQUENTIAL
                ;1774   ; THESE ARE THE DATA PATH CONTROL ROM CONTENTS
                ;1775   ;
                ;1776   ; THE FOLLOWING LOCATIONS ARE USED BY THE DECODE.IS MICRO INSTRUCTION
                ;1777   ;
                ;1778   ; THIS INSTRUCTION WILL DO PC+1 OPERATIONS
                ;1779   ;
                ;1780   ;               BACK UP PC IN 2901 RAM
                ;1781   000:
                ;1782   DECODE.IS:
C 0000, 3D8     ;1783           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0001, 3D8     ;1784           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0002, 3D8     ;1785           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0003, 3D8     ;1786           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0004, 3D8     ;1787           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0005, 3D8     ;1788           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0006, 3D8     ;1789           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0007, 3D8     ;1790           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0008, 3D8     ;1791           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0009, 3D8     ;1792           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 000A, 3D8     ;1793           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 000B, 3D8     ;1794           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 000C, 3D8     ;1795           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 000D, 3D8     ;1796           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 000E, 3D8     ;1797           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 000F, 3D8     ;1798           SRC/D.0,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
                ;1799   ;
                ;1800   ;               DON'T BACK UP PC
                ;1801   ;
C 0010, 3C8     ;1802           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0011, 3C8     ;1803           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0012, 3C8     ;1804           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0013, 3C8     ;1805           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0014, 3C8     ;1806           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0015, 3C8     ;1807           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0016, 3C8     ;1808           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0017, 3C8     ;1809           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0018, 3C8     ;1810           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 0019, 3C8     ;1811           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 001A, 3C8     ;1812           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 001B, 3C8     ;1813           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 001C, 3C8     ;1814           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 001D, 3C8     ;1815           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 001E, 3C8     ;1816           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 001F, 3C8     ;1817           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
```

```
                ;1818   .PAGE
                ;1819   ;  THESE ADDRESSES ARE USED BY THE 'JUMP' MICRO INSTRUCTION
                ;1820   ;
                ;1821   ; THE FOLLOWING CONTROL CODES WILL INSURE THAT NO DATA IS DESTROYED WITHIN
                ;1822   ; THE 2901 DURING JUMP OR JSR OPERATIONS.
                ;1823   ;
                ;1824   020:
                ;1825   JUMP:
C 0020, 3CB     ;1826           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0021, 3CB     ;1827           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0022, 3CB     ;1828           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0023, 3CB     ;1829           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0024, 3CB     ;1830           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0025, 3CB     ;1831           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0026, 3CB     ;1832           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0027, 3CB     ;1833           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0028, 3CB     ;1834           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0029, 3CB     ;1835           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 002A, 3CB     ;1836           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 002B, 3CB     ;1837           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 002C, 3CB     ;1838           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 002D, 3CB     ;1839           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 002E, 3CB     ;1840           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 002F, 3CB     ;1841           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0030, 3CB     ;1842           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0031, 3CB     ;1843           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0032, 3CB     ;1844           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0033, 3CB     ;1845           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0034, 3CB     ;1846           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0035, 3CB     ;1847           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0036, 3CB     ;1848           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0037, 3CB     ;1849           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0038, 3CB     ;1850           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 0039, 3CB     ;1851           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 003A, 3CB     ;1852           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 003B, 3CB     ;1853           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 003C, 3CB     ;1854           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 003D, 3CB     ;1855           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 003E, 3CB     ;1856           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
C 003F, 3CB     ;1857           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/CIN
```

```
                    ;1858   .PAGE
                    ;1859   ; THESE LOCATIONS ARE USED BY THE MISC' MICRO INSTRUCTION
                    ;1860   ;
                    ;1861   ; THE FOLLOWING CONTROL CODES WILL INSURE THAT NO DATA IS DESTROYED WITHIN
                    ;1862   ; THE 2901 DURING MISC INSTRUCTION EXECUTION.
                    ;1863   ;
                    ;1864   040:
                    ;1865   MISC:
C 0040, 313         ;1866           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0041, 313         ;1867           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0042, 313         ;1868           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0043, 313         ;1869           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0044, 313         ;1870           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0045, 313         ;1871           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0046, 313         ;1872           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0047, 313         ;1873           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0048, 313         ;1874           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0049, 313         ;1875           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 004A, 313         ;1876           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 004B, 313         ;1877           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 004C, 313         ;1878           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 004D, 313         ;1879           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 004E, 313         ;1880           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 004F, 313         ;1881           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0050, 313         ;1882           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0051, 313         ;1883           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0052, 313         ;1884           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0053, 313         ;1885           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0054, 313         ;1886           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0055, 313         ;1887           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0056, 313         ;1888           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0057, 313         ;1889           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0058, 313         ;1890           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0059, 313         ;1891           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 005A, 313         ;1892           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 005B, 313         ;1893           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 005C, 313         ;1894           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 005D, 313         ;1895           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 005E, 313         ;1896           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 005F, 313         ;1897           SRC/0.A,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
```

```
                    ;1898   .PAGE
                    ;1899   ; THESE ADDRESSES ARE USED BY THE MEM.REQ MICRO INSTRUCTION
                    ;1900   ;
                    ;1901   ; THIS INSTRUCTION CAUSES WR[5] TO BE LOADED WITH THE VIRTUAL ADDRESS
                    ;1902   ; OF THE MEMORY REFERENCE.
                    ;1903   060:
                    ;1904   MEM.REQ:
C 0060, 3DB         ;1905           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0061, 3DB         ;1906           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0062, 3DB         ;1907           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0063, 3DB         ;1908           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0064, 3DB         ;1909           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0065, 3DB         ;1910           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0066, 3DB         ;1911           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0067, 3DB         ;1912           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0068, 3DB         ;1913           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0069, 3DB         ;1914           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 006A, 3DB         ;1915           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 006B, 3DB         ;1916           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 006C, 3DB         ;1917           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 006D, 3DB         ;1918           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 006E, 3DB         ;1919           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 006F, 3DB         ;1920           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0070, 3DB         ;1921           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0071, 3DB         ;1922           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0072, 3DB         ;1923           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0073, 3DB         ;1924           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0074, 3DB         ;1925           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0075, 3DB         ;1926           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0076, 3DB         ;1927           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0077, 3DB         ;1928           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0078, 3DB         ;1929           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 0079, 3DB         ;1930           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 007A, 3DB         ;1931           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 007B, 3DB         ;1932           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 007C, 3DB         ;1933           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 007D, 3DB         ;1934           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 007E, 3DB         ;1935           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 007F, 3DB         ;1936           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
```

```
                ;1937   .PAGE
                ;1938   ; THESE ARE THE ADDRESSES FOR THE EXTENDED MICRO INSTRUCTION
                ;1939   ; ADDRESSES START AT 80-FF
                ;1940   080:
                ;1941
                ;1942   WR.PLUS.0.TO.WR:
C 0080, 118     ;1943           SRC/0.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
                ;1944   WR.INC.WR:
C 0081, 318     ;1945           SRC/0.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
                ;1946   WR.NEG.WR:
C 0082, 31A     ;1947           SRC/0.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
                ;1948   WR.COM.WR:
C 0083, 31F     ;1949           SRC/0.A,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
                ;1950
                ;1951   WR.DEC.WR:
C 0084, 119     ;1952           SRC/0.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
                ;1953   FILL85:
C 0085, 108     ;1954           SRC/0.A
                ;1955   MOV.Q.WR:
C 0086, 298     ;1956           SRC/0.Q,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
                ;1957   FILL87:
C 0087, 088     ;1958           SRC/0.Q
                ;1959
                ;1960   COND.ADD&SHIFT:
C 0088, 060     ;1961           SRC/A.B,ALU/R.PLUS.S,DST/RSHF.RAM.Q,CIN/NO.CIN
                ;1962   COND.SUB&SHIFT:
C 0089, 261     ;1963           SRC/A.B,ALU/S.MINUS.R,DST/RSHF.RAM.Q,CIN/CIN
                ;1964   FILL8A:
C 008A, 04B     ;1965           SRC/A.B
                ;1966   SHL2:
C 008B, 078     ;1967           SRC/A.B,ALU/R.PLUS.S,DST/LSHF.RAM,CIN/NO.CIN
                ;1968
                ;1969   ASHRC:
C 008C, 0E3     ;1970           SRC/0.B,ALU/R.OR.S,DST/RSHF.RAM.Q,CIN/NO.CIN
                ;1971   ASHR:
C 008D, 2EB     ;1972           SRC/0.B,ALU/R.OR.S,DST/RSHF.RAM,CIN/CIN
                ;1973   ASHLC:
C 008E, 2F3     ;1974           SRC/0.B,ALU/R.OR.S,DST/LSHF.RAM.Q,CIN/CIN
                ;1975   ASHL:
C 008F, 2FB     ;1976           SRC/0.B,ALU/R.OR.S,DST/LSHF.RAM,CIN/CIN
                ;1977
                ;1978   Q.PLUS.0:
C 0090, 088     ;1979           SRC/0.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
                ;1980   FILL91:
C 0091, 08B     ;1981           SRC/0.Q
                ;1982   WR.CMP.Q:
C 0092, 20A     ;1983           SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
                ;1984   Q.CMP.WR:
C 0093, 209     ;1985           SRC/A.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
                ;1986
                ;1987   DEC.Q:
C 0094, 081     ;1988           SRC/0.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/NO.CIN
                ;1989   INC.Q:
C 0095, 280     ;1990           SRC/0.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/CIN
                ;1991   NEG.Q:
```

```
C 0096, 282     ;1992           SRC/0.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
                ;1993   COM.Q:
C 0097, 287     ;1994           SRC/0.Q,ALU/R.XNOR.S,DST/LOAD.Q,CIN/CIN
                ;1995
                ;1996   WR.BIT.WR:
C 0098, 04C     ;1997           SRC/A.B,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
                ;1998   WR.CMP.WR:
C 0099, 24A     ;1999           SRC/A.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
                ;2000   FILL9A:
C 009A, 04B     ;2001           SRC/A.B
                ;2002   WR.PLUS.WR+1:
C 009B, 258     ;2003           SRC/A.B,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
                ;2004
                ;2005   FILL9C:
C 009C, 04B     ;2006           SRC/A.B
                ;2007   FILL9D:
C 009D, 04B     ;2008           SRC/A.B
                ;2009   FILL9E:
C 009E, 0CB     ;2010           SRC/0.B
                ;2011   FILL9F:
C 009F, 0CB     ;2012           SRC/0.B
                ;2013
                ;2014   WR.PLUS.Q:
C 00A0, 000     ;2015           SRC/A.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
                ;2016   WR.FROM.Q:
C 00A1, 201     ;2017           SRC/A.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
                ;2018   Q.FROM.WR.Q:
C 00A2, 202     ;2019           SRC/A.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
                ;2020   WR.OR.Q:
C 00A3, 203     ;2021           SRC/A.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
                ;2022
                ;2023   WR.AND.Q:
C 00A4, 004     ;2024           SRC/A.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
                ;2025   WR.MASK.Q:
C 00A5, 205     ;2026           SRC/A.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/CIN
                ;2027   WR.XOR.Q:
C 00A6, 206     ;2028           SRC/A.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
                ;2029   FILLA7:
C 00A7, 00B     ;2030           SRC/A.Q
                ;2031
                ;2032   WR.PLUS.WR.Q:
C 00A8, 040     ;2033           SRC/A.B,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
                ;2034   WR.FROM.WR.Q:
C 00A9, 241     ;2035           SRC/A.B,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
                ;2036   FILLAA:
C 00AA, 04B     ;2037           SRC/A.B
                ;2038   WR.OR.WR.Q:
C 00AB, 243     ;2039           SRC/A.B,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
                ;2040
                ;2041   WR.AND.WR.Q:
C 00AC, 044     ;2042           SRC/A.B,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
                ;2043   WR.MASK.WR.Q:
C 00AD, 245     ;2044           SRC/A.B,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/CIN
                ;2045   WR.XOR.WR.Q:
C 00AE, 246     ;2046           SRC/A.B,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
```

```
                    ;2047   FILLAF:
C 00AF, 04B         ;2048           SRC/A.B
                    ;2049
                    ;2050   Q.PLUS.WR:
C 00B0, 018         ;2051           SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
                    ;2052   WR.FROM.Q.WR:
C 00B1, 219         ;2053           SRC/A.Q,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
                    ;2054   Q.FROM.WR:
C 00B2, 21A         ;2055           SRC/A.Q,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
                    ;2056   Q.OR.WR:
C 00B3, 21B         ;2057           SRC/A.Q,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
                    ;2058
                    ;2059   Q.AND.WR:
C 00B4, 01C         ;2060           SRC/A.Q,ALU/R.AND.S,DST/WRITE.B.F,CIN/NO.CIN
                    ;2061   FILLB5:
C 00B5, 00B         ;2062           SRC/A.Q
                    ;2063   Q.XOR.WR:
C 00B6, 21E         ;2064           SRC/A.Q,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
                    ;2065   Q.BIT.WR:
C 00B7, 20C         ;2066           SRC/A.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
                    ;2067
                    ;2068   WR.PLUS.WR:
C 00B8, 058         ;2069           SRC/A.B,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
                    ;2070   WR.FROM.WR:
C 00B9, 259         ;2071           SRC/A.B,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
                    ;2072   WR.FROM.WR.WR:
C 00BA, 25A         ;2073           SRC/A.B,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
                    ;2074   WR.OR.WR:
C 00BB, 25B         ;2075           SRC/A.B,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
                    ;2076
                    ;2077   WR.FROM.WR-1:
C 00BC, 059         ;2078           SRC/A.B,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
                    ;2079   WR.MASK.WR:
C 00BD, 25D         ;2080           SRC/A.B,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
                    ;2081   WR.XOR.WR:
C 00BE, 25E         ;2082           SRC/A.B,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
                    ;2083   WR.AND.WR:
C 00BF, 25C         ;2084           SRC/A.B,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
                    ;2085
```

```
                    ;2086   .PAGE
                    ;2087   ; THIS IS THE DP CODES FOR THE 'MOVE' MICRO INSTRUCTION
                    ;2088   ;
                    ;2089   0C0:
                    ;2090
                    ;2091   MOV.MEM.WR:
C 00C0, 1D3         ;2092           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C1, 1D3         ;2093           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C2, 1D3         ;2094           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C3, 1D3         ;2095           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C4, 1D3         ;2096           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C5, 1D3         ;2097           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C6, 1D3         ;2098           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 00C7, 1D3         ;2099           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
                    ;2100
                    ;2101   MOV.LS.MEM:
C 00C8, 1CB         ;2102           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00C9, 1CB         ;2103           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00CA, 1CB         ;2104           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00CB, 1CB         ;2105           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00CC, 1CB         ;2106           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00CD, 1CB         ;2107           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00CE, 1CB         ;2108           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00CF, 1CB         ;2109           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
                    ;2110
                    ;2111   MOV.XWR.Q:
C 00D0, 0C3         ;2112           SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D1, 0C3         ;2113           SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D2, 0C3         ;2114           SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D3, 0C3         ;2115           SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D4, 0C3         ;2116           SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D5, 0C3         ;2117           SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D6, 0C3         ;2118           SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 00D7, 0C3         ;2119           SRC/0.B,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
                    ;2120
                    ;2121   MOV.LS.WR:
C 00D8, 1DB         ;2122           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00D9, 1DB         ;2123           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00DA, 1DB         ;2124           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00DB, 1DB         ;2125           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00DC, 1DB         ;2126           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00DD, 1DB         ;2127           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00DE, 1DB         ;2128           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
C 00DF, 1DB         ;2129           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.F,CIN/NO.CIN
                    ;2130
```

```
                    ;2131   .PAGE
                    ;2132   MOV.MEM.LS:
C 00E0, 1CB         ;2133           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E1, 1CB         ;2134           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E2, 1CB         ;2135           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E3, 1CB         ;2136           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E4, 1CB         ;2137           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E5, 1CB         ;2138           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E6, 1CB         ;2139           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E7, 1CB         ;2140           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
                    ;2141   MOV.ACC.LS:
C 00E8, 1CB         ;2142           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00E9, 1CB         ;2143           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00EA, 1CB         ;2144           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00EB, 1CB         ;2145           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00EC, 1CB         ;2146           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00ED, 1CB         ;2147           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00EE, 1CB         ;2148           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00EF, 1CB         ;2149           SRC/D.0,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
                    ;2150
                    ;2151
                    ;2152   WR.PLUS.OS:
C 00F0, 148         ;2153           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F1, 148         ;2154           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F2, 148         ;2155           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F3, 148         ;2156           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F4, 148         ;2157           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F5, 148         ;2158           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F6, 148         ;2159           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 00F7, 148         ;2160           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
                    ;2161
                    ;2162   MOV.WR.LS:
C 00F8, 10B         ;2163           SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00F9, 10B         ;2164           SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00FA, 10B         ;2165           SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00FB, 10B         ;2166           SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00FC, 10B         ;2167           SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00FD, 10B         ;2168           SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00FE, 10B         ;216,           SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
C 00FF, 10B         ;2170           SRC/0.A,ALU/R.OR.S,DST/NOP,CIN/NO.CIN
                    ;2171
```

```
                    ;2172   .PAGE
                    ;2173   ; THIS GROUP OF INSTRUCTIONS IS FOR THE 'BASIC' MICRO INSTRUCTION
                    ;2174   ; THIS USES ADDRESSES 100-1FF
                    ;2175
                    ;2176   100:
                    ;2177   LS.PLUS.WR:
C 0100, 158         ;2178           SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
C 0101, 158         ;2179           SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
C 0102, 158         ;2180           SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
C 0103, 158         ;2181           SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
                    ;2182   LS.FROM.WR:
C 0104, 359         ;2183           SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
C 0105, 359         ;2184           SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
C 0106, 359         ;2185           SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
C 0107, 359         ;2186           SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
                    ;2187   LS.AND.WR:
C 0108, 35C         ;2188           SRC/D.A,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
C 0109, 35C         ;2189           SRC/D.A,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
C 010A, 35C         ;2190           SRC/D.A,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
C 010B, 35C         ;2191           SRC/D.A,ALU/R.AND.S,DST/WRITE.B.F,CIN/CIN
                    ;2192   LS.XOR.WR:
C 010C, 35E         ;2193           SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
C 010D, 35E         ;2194           SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
C 010E, 35E         ;2195           SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
C 010F, 35E         ;2196           SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.F,CIN/CIN
                    ;2197
                    ;2198   LS.FROM.WR-1:
C 0110, 159         ;2199           SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
C 0111, 159         ;2200           SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
C 0112, 159         ;2201           SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
C 0113, 159         ;2202           SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/NO.CIN
                    ;2203   LS.MASK.WR:
C 0114, 35D         ;2204           SRC/D.A,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
C 0115, 35D         ;2205           SRC/D.A,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
C 0116, 35D         ;2206           SRC/D.A,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
C 0117, 35D         ;2207           SRC/D.A,ALU/NOTR.AND.S,DST/WRITE.B.F,CIN/CIN
                    ;2208   LS.PLUS.WR+1:
C 0118, 358         ;2209           SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 0119, 358         ;2210           SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 011A, 358         ;2211           SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
C 011B, 358         ;2212           SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/CIN
                    ;2213   LS.OR.WR:
C 011C, 35B         ;2214           SRC/D.A,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 011D, 35B         ;2215           SRC/D.A,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 011E, 35B         ;2216           SRC/D.A,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
C 011F, 35B         ;2217           SRC/D.A,ALU/R.OR.S,DST/WRITE.B.F,CIN/CIN
                    ;2218
                    ;2219   WR.PLUS.LS.Q:
C 0120, 140         ;2220           SRC/D.A,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
C 0121, 140         ;2221           SRC/D.A,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
C 0122, 140         ;2222           SRC/D.A,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
C 0123, 140         ;2223           SRC/D.A,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
                    ;2224   LS.FROM.WR.Q:
C 0124, 341         ;2225           SRC/D.A,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
C 0125, 341         ;2226           SRC/D.A,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
```

```
C 0126, 341     ;2227           SRC/D.A,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
C 0127, 341     ;2228           SRC/D.A,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
                ;2229   WR.FROM.LS.Q:
C 0128, 342     ;2230           SRC/D.A,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
C 0129, 342     ;2231           SRC/D.A,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
C 012A, 342     ;2232           SRC/D.A,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
C 012B, 342     ;2233           SRC/D.A,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
                ;2234   WR.OR.LS.Q:
C 012C, 343     ;2235           SRC/D.A,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
C 012D, 343     ;2236           SRC/D.A,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
C 012E, 343     ;2237           SRC/D.A,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
C 012F, 343     ;2238           SRC/D.A,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
                ;2239
                ;2240   WR.AND.LS.Q:
C 0130, 144     ;2241           SRC/D.A,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
C 0131, 144     ;2242           SRC/D.A,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
C 0132, 144     ;2243           SRC/D.A,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
C 0133, 144     ;2244           SRC/D.A,ALU/R.AND.S,DST/LOAD.Q,CIN/NO.CIN
                ;2245   WR.XOR.LS.Q:
C 0134, 346     ;2246           SRC/D.A,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
C 0135, 346     ;2247           SRC/D.A,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
C 0136, 346     ;2248           SRC/D.A,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
C 0137, 346     ;2249           SRC/D.A,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
                ;2250   LS.CMP.WR:
C 0138, 34A     ;2251           SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 0139, 34A     ;2252           SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 013A, 34A     ;2253           SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 013B, 34A     ;2254           SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
                ;2255   WR.CMP.LS:
C 013C, 349     ;2256           SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 013D, 349     ;2257           SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 013E, 349     ;2258           SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 013F, 349     ;2259           SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
                ;2260
                ;2261   LS.PLUS.Q:
C 0140, 180     ;2262           SRC/D.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
C 0141, 180     ;2263           SRC/D.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
C 0142, 180     ;2264           SRC/D.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
C 0143, 180     ;2265           SRC/D.Q,ALU/R.PLUS.S,DST/LOAD.Q,CIN/NO.CIN
                ;2266   LS.FROM.Q:
C 0144, 381     ;2267           SRC/D.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
C 0145, 381     ;2268           SRC/D.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
C 0146, 381     ;2269           SRC/D.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
C 0147, 381     ;2270           SRC/D.Q,ALU/S.MINUS.R,DST/LOAD.Q,CIN/CIN
                ;2271   Q.FROM.LS.Q:
C 0148, 382     ;2272           SRC/D.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
C 0149, 382     ;2273           SRC/D.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
C 014A, 382     ;2274           SRC/D.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
C 014B, 382     ;2275           SRC/D.Q,ALU/R.MINUS.S,DST/LOAD.Q,CIN/CIN
                ;2276   LS.OR.Q:
C 014C, 383     ;2277           SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
C 014D, 383     ;2278           SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
C 014E, 383     ;2279           SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
C 014F, 383     ;2280           SRC/D.Q,ALU/R.OR.S,DST/LOAD.Q,CIN/CIN
                ;2281
```

```
                ;2282   LS.MASK.Q:
C 0150, 185     ;2283           SRC/D.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/NO.CIN
C 0151, 185     ;2284           SRC/D.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/NO.CIN
C 0152, 185     ;2285           SRC/D.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/NO.CIN
C 0153, 185     ;2286           SRC/D.Q,ALU/NOTR.AND.S,DST/LOAD.Q,CIN/NO.CIN
                ;2287   LS.XOR.Q:
C 0154, 386     ;2288           SRC/D.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
C 0155, 386     ;2289           SRC/D.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
C 0156, 386     ;2290           SRC/D.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
C 0157, 386     ;2291           SRC/D.Q,ALU/R.XOR.S,DST/LOAD.Q,CIN/CIN
                ;2292   LS.AND.Q:
C 0158, 384     ;2293           SRC/D.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/CIN
C 0159, 384     ;2294           SRC/D.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/CIN
C 015A, 384     ;2295           SRC/D.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/CIN
C 015B, 384     ;2296           SRC/D.Q,ALU/R.AND.S,DST/LOAD.Q,CIN/CIN
                ;2297   LS.BIT.Q:
C 015C, 38C     ;2298           SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
C 015D, 38C     ;2299           SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
C 015E, 38C     ;2300           SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
C 015F, 38C     ;2301           SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/CIN
                ;2302
                ;2303   MOV.LS.Q:
C 0160, 1C3     ;2304           SRC/D.0,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 0161, 1C3     ;2305           SRC/D.0,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 0162, 1C3     ;2306           SRC/D.0,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
C 0163, 1C3     ;2307           SRC/D.0,ALU/R.OR.S,DST/LOAD.Q,CIN/NO.CIN
                ;2308   WR.BIT.LS:
C 0164, 34C     ;2309           SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
C 0165, 34C     ;2310           SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
C 0166, 34C     ;2311           SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
C 0167, 34C     ;2312           SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
                ;2313   LS.CMP.Q:
C 0168, 38A     ;2314           SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 0169, 38A     ;2315           SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 016A, 38A     ;2316           SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 016B, 38A     ;2317           SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
                ;2318   Q.CMP.LS:
C 016C, 389     ;2319           SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 016D, 389     ;2320           SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 016E, 389     ;2321           SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 016F, 389     ;2322           SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
                ;2323
                ;2324   Q.PLUS.LS.TO.WR:
C 0170, 198     ;2325           SRC/D.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
C 0171, 198     ;2326           SRC/D.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
C 0172, 198     ;2327           SRC/D.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
C 0173, 198     ;2328           SRC/D.Q,ALU/R.PLUS.S,DST/WRITE.B.F,CIN/NO.CIN
                ;2329   WRA.FROM.LS.WRB:
C 0174, 35A     ;2330           SRC/D.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
C 0175, 35A     ;2331           SRC/D.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
C 0176, 35A     ;2332           SRC/D.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
C 0177, 35A     ;2333           SRC/D.A,ALU/R.MINUS.S,DST/WRITE.B.F,CIN/CIN
                ;2334   LS.NEG.WR:
C 0178, 3D9     ;2335           SRC/D.0,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
C 0179, 3D9     ;2336           SRC/D.0,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
```

```
C 017A, 3D9     ;2337           SRC/D.0,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
C 017B, 3D9     ;2338           SRC/D.0,ALU/S.MINUS.R,DST/WRITE.B.F,CIN/CIN
                ;2339   LS.COM.WR:
C 017C, 3DF     ;2340           SRC/D.0,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
C 017D, 3DF     ;2341           SRC/D.0,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
C 017E, 3DF     ;2342           SRC/D.0,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
C 017F, 3DF     ;2343           SRC/D.0,ALU/R.XNOR.S,DST/WRITE.B.F,CIN/CIN
```

```
                    ;2344   .PAGE
                    ;2345   ;
                    ;2346   ; THE FOLLOWING LOCATIONS HAVE THE LOCAL STORE AS THEIR DESTINATION.
                    ;2347   ;
                    ;2348   ; THIS FACT IS USED BY HARDWARE TO GENERATE LOCAL STORE WRITE  PULSES
                    ;2349
                    ;2350   180:
                    ;2351
                    ;2352   LS.PLUS.WR.XCHG:
C 0180, 150         ;2353           SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
C 0181, 150         ;2354           SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
C 0182, 150         ;2355           SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
C 0183, 150         ;2356           SRC/D.A,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
                    ;2357   LS.FROM.WR.XCHG:
C 0184, 351         ;2358           SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.A,CIN/CIN
C 0185, 351         ;2359           SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.A,CIN/CIN
C 0186, 351         ;2360           SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.A,CIN/CIN
C 0187, 351         ;2361           SRC/D.A,ALU/S.MINUS.R,DST/WRITE.B.A,CIN/CIN
                    ;2362   LS.AND.WR.XCHG:
C 0188, 354         ;2363           SRC/D.A,ALU/R.AND.S,DST/WRITE.B.A,CIN/CIN
C 0189, 354         ;2364           SRC/D.A,ALU/R.AND.S,DST/WRITE.B.A,CIN/CIN
C 018A, 354         ;2365           SRC/D.A,ALU/R.AND.S,DST/WRITE.B.A,CIN/CIN
C 018B, 354         ;2366           SRC/D.A,ALU/R.AND.S,DST/WRITE.B.A,CIN/CIN
                    ;2367   LS.XOR.WR.XCHG:
C 018C, 356         ;2368           SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.A,CIN/CIN
C 018D, 356         ;2369           SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.A,CIN/CIN
C 018E, 356         ;2370           SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.A,CIN/CIN
C 018F, 356         ;2371           SRC/D.A,ALU/R.XOR.S,DST/WRITE.B.A,CIN/CIN
                    ;2372
                    ;2373   SWAP.LS.WR:
C 0190, 1D3         ;2374           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 0191, 1D3         ;2375           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 0192, 1D3         ;2376           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
C 0193, 1D3         ;2377           SRC/D.0,ALU/R.OR.S,DST/WRITE.B.A,CIN/NO.CIN
                    ;2378   CLR.LS:
C 0194, 3CC         ;2379           SRC/D.0,ALU/R.AND.S,DST/NOP,CIN/CIN
C 0195, 3CC         ;2380           SRC/D.0,ALU/R.AND.S,DST/NOP,CIN/CIN
C 0196, 3CC         ;2381           SRC/D.0,ALU/R.AND.S,DST/NOP,CIN/CIN
C 0197, 3CC         ;2382           SRC/D.0,ALU/R.AND.S,DST/NOP,CIN/CIN
                    ;2383   MOV.Q.WR&WR.LS:
C 0198, 293         ;2384           SRC/0.Q,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 0199, 293         ;2385           SRC/0.Q,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 019A, 293         ;2386           SRC/0.Q,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
C 019B, 293         ;2387           SRC/0.Q,ALU/R.OR.S,DST/WRITE.B.A,CIN/CIN
                    ;2388   WR.PLUS.Q.XCHG:
C 019C, 010         ;2389           SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
C 019D, 010         ;2390           SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
C 019E, 010         ;2391           SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
C 019F, 010         ;2392           SRC/A.Q,ALU/R.PLUS.S,DST/WRITE.B.A,CIN/NO.CIN
                    ;2393
                    ;2394   WR.FROM.LS-1:
C 01A0, 14A         ;2395           SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
C 01A1, 14A         ;2396           SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
C 01A2, 14A         ;2397           SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
C 01A3, 14A         ;2398           SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
```

```
                    ;2399   LS.FROM.WR.LS:
C 01A4, 349         ;2400           SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01A5, 349         ;2401           SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01A6, 349         ;2402           SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01A7, 349         ;2403           SRC/D.A,ALU/S.MINUS.R,DST/NOP,CIN/CIN
                    ;2404   WR.PLUS.LS+1:
C 01A8, 348         ;2405           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 01A9, 348         ;2406           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 01AA, 348         ;2407           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 01AB, 348         ;2408           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/CIN
                    ;2409   WR.FROM.LS:
C 01AC, 34A         ;2410           SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01AD, 34A         ;2411           SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01AE, 34A         ;2412           SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01AF, 34A         ;2413           SRC/D.A,ALU/R.MINUS.S,DST/NOP,CIN/CIN
                    ;2414
                    ;2415   WR.PLUS.LS:
C 01B0, 148         ;2416           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01B1, 148         ;2417           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01B2, 148         ;2418           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01B3, 148         ;2419           SRC/D.A,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
                    ;2420   WR.OR.LS:
C 01B4, 34B         ;2421           SRC/D.A,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01B5, 34B         ;2422           SRC/D.A,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01B6, 34B         ;2423           SRC/D.A,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01B7, 34B         ;2424           SRC/D.A,ALU/R.OR.S,DST/NOP,CIN/CIN
                    ;2425   WR.AND.LS:
C 01B8, 34C         ;2426           SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
C 01B9, 34C         ;2427           SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
C 01BA, 34C         ;2428           SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
C 01BB, 34C         ;2429           SRC/D.A,ALU/R.AND.S,DST/NOP,CIN/CIN
                    ;2430   WR.XOR.LS:
C 01BC, 34E         ;2431           SRC/D.A,ALU/R.XOR.S,DST/NOP,CIN/CIN
C 01BD, 34E         ;2432           SRC/D.A,ALU/R.XOR.S,DST/NOP,CIN/CIN
C 01BE, 34E         ;2433           SRC/D.A,ALU/R.XOR.S,DST/NOP,CIN/CIN
C 01BF, 34E         ;2434           SRC/D.A,ALU/R.XOR.S,DST/NOP,CIN/CIN
                    ;2435
                    ;2436   Q.PLUS.LS:
C 01C0, 188         ;2437           SRC/D.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01C1, 188         ;2438           SRC/D.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01C2, 188         ;2439           SRC/D.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01C3, 188         ;2440           SRC/D.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
                    ;2441   LS.FROM.Q.LS:
C 01C4, 389         ;2442           SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01C5, 389         ;2443           SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01C6, 389         ;2444           SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01C7, 389         ;2445           SRC/D.Q,ALU/S.MINUS.R,DST/NOP,CIN/CIN
                    ;2446   Q.FROM.LS:
C 01C8, 38A         ;2447           SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01C9, 38A         ;2448           SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01CA, 38A         ;2449           SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01CB, 38A         ;2450           SRC/D.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
                    ;2451   Q.OR.LS:
C 01CC, 38B         ;2452           SRC/D.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01CD, 38B         ;2453           SRC/D.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
```

```
C 01CE, 38B     ;2454           SRC/D.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01CF, 38B     ;2455           SRC/D.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
                ;2456
                ;2457   Q.AND.LS:
C 01D0, 18C     ;2458           SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
C 01D1, 18C     ;2459           SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
C 01D2, 18C     ;2460           SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
C 01D3, 18C     ;2461           SRC/D.Q,ALU/R.AND.S,DST/NOP,CIN/NO.CIN
                ;2462   Q.XOR.LS:
C 01D4, 38E     ;2463           SRC/D.Q,ALU/R.XOR.S,DST/NOP,CIN/CIN
C 01D5, 38E     ;2464           SRC/D.Q,ALU/R.XOR.S,DST/NOP,CIN/CIN
C 01D6, 38E     ;2465           SRC/D.Q,ALU/R.XOR.S,DST/NOP,CIN/CIN
C 01D7, 38E     ;2466           SRC/D.Q,ALU/R.XOR.S,DST/NOP,CIN/CIN
                ;2467   MOV.Q.LS:
C 01D8, 28B     ;2468           SRC/0.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01D9, 28B     ;2469           SRC/0.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01DA, 28B     ;2470           SRC/0.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
C 01DB, 28B     ;2471           SRC/0.Q,ALU/R.OR.S,DST/NOP,CIN/CIN
                ;2472   Q.NEG.LS:
C 01DC, 28A     ;2473           SRC/0.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01DD, 28A     ;2474           SRC/0.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01DE, 28A     ;2475           SRC/0.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01DF, 28A     ;2476           SRC/0.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
                ;2477
                ;2478   WR.COM.LS:
C 01E0, 0CF     ;2479           SRC/0.B,ALU/R.XNOR.S,DST/NOP,CIN/NO.CIN
C 01E1, 0CF     ;2480           SRC/0.B,ALU/R.XNOR.S,DST/NOP,CIN/NO.CIN
C 01E2, 0CF     ;2481           SRC/0.B,ALU/R.XNOR.S,DST/NOP,CIN/NO.CIN
C 01E3, 0CF     ;2482           SRC/0.B,ALU/R.XNOR.S,DST/NOP,CIN/NO.CIN
                ;2483   WR.NEG.LS:
C 01E4, 2CA     ;2484           SRC/0.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01E5, 2CA     ;2485           SRC/0.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01E6, 2CA     ;2486           SRC/0.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01E7, 2CA     ;2487           SRC/0.B,ALU/R.MINUS.S,DST/NOP,CIN/CIN
                ;2488   Q.PLUS.WR.TO.LS:
C 01E8, 008     ;2489           SRC/A.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01E9, 008     ;2490           SRC/A.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01EA, 008     ;2491           SRC/A.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
C 01EB, 008     ;2492           SRC/A.Q,ALU/R.PLUS.S,DST/NOP,CIN/NO.CIN
                ;2493   Q.FROM.WR.TO.LS:
C 01EC, 20A     ;2494           SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01ED, 20A     ;2495           SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01EE, 20A     ;2496           SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
C 01EF, 20A     ;2497           SRC/A.Q,ALU/R.MINUS.S,DST/NOP,CIN/CIN
                ;2498
                ;2499   DEC.LS:
C 01F0, 1CA     ;2500           SRC/D.0,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
C 01F1, 1CA     ;2501           SRC/D.0,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
C 01F2, 1CA     ;2502           SRC/D.0,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
C 01F3, 1CA     ;2503           SRC/D.0,ALU/R.MINUS.S,DST/NOP,CIN/NO.CIN
                ;2504   COM.LS:
C 01F4, 3CF     ;2505           SRC/D.0,ALU/R.XNOR.S,DST/NOP,CIN/CIN
C 01F5, 3CF     ;2506           SRC/D.0,ALU/R.XNOR.S,DST/NOP,CIN/CIN
C 01F6, 3CF     ;2507           SRC/D.0,ALU/R.XNOR.S,DST/NOP,CIN/CIN
C 01F7, 3CF     ;2508           SRC/D.0,ALU/R.XNOR.S,DST/NOP,CIN/CIN
```

```
                    ;2509   NEG.LS:
C 01F8, 3C9         ;2510           SRC/D.0,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01F9, 3C9         ;2511           SRC/D.0,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01FA, 3C9         ;2512           SRC/D.0,ALU/S.MINUS.R,DST/NOP,CIN/CIN
C 01FB, 3C9         ;2513           SRC/D.0,ALU/S.MINUS.R,DST/NOP,CIN/CIN
                    ;2514   INC.LS:
C 01FC, 3C8         ;2515           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 01FD, 3C8         ;2516           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 01FE, 3C8         ;2517           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
C 01FF, 3C8         ;2518           SRC/D.0,ALU/R.PLUS.S,DST/NOP,CIN/CIN
                    ;2519
                    ;2520   .UCODE
```

```
;2521   .PAGE   'DIAGNOSTIC MACROS AND DEFINITIONS'
;2522
;2523   D.ADRS/=<15:9>,.VALIDITY=<D.VAL>,.DEFAULT=0
;2524
;2525
;2526           SAV.WR0=1               ; STORAGE FOR WR0
;2527           SAV.WR1=2               ; STORAGE FOR WR1
;2528           SAV.WR2=3               ; STORAGE FOR WR2
;2529           SAV.WR3=4               ; STORAGE FOR WR3
;2530           T5.SUB=5                ; RESERVED FOR COMMON SUBROUTINES
;2531           T6.SUB=6                ; RESERVED FOR COMMON SUBROUTINES
;2532           T7.SUB=7                ; RESERVED FOR COMMON SUBROUTINES
;2533           TRANSFER.POINTER=7      ; POINTER FOR TRANSFER ROUTINE
;2534           MEMORY.SIZE=7           ; STORAGE FOR MEMORY SIZE IN 256KB BLOCKS AFTER
;2535                                   ;  SIZING
;2536           FPA.FOUND=7             ; STORAGE FOR RESULT OF SIZING FOR FPA PRESENT
;2537           UBE.FOUND=7             ; STORAGE FOR RESULT OF SIZING FOR UBE PRESENT
;2538           T8=8                    ; TEMP LOCATION 8
;2539           T9=9                    ; TEMP LOCATION 9
;2540           T10=0A                  ; TEMP LOCATION 10
;2541           T11=0B                  ; TEMP LOCATION 11
;2542           T12=0C                  ; TEMP LOCATION 12
;2543           T13=0D                  ; TEMP LOCATION 13
;2544           T14=0E                  ; TEMP LOCATION 14
;2545           T15=0F                  ; TEMP LOCATION 15
;2546           PC=10                   ; MACRO PROGRAM COUNTER
;2547
;2548           WR.BACKUP=11            ; BACKUP WR FOR MOV MEM.DATA TO WR[]
;2549           MM.LASTADDR+1=12        ; STORAGE OF LAST ADDRESS IN MAIN MEMORY PLUS
;2550                                   ;  1 (FIRST NON-EXISTANT MEMORY ADDRESS)
;2551           #FF=13                  ; MASK
;2552           #FFFF=14                ; MASK
;2553           #FF000000=15            ; MASK
;2554           #FFFFFF00=16            ; MASK
;2555           CSR0.MASK=16            ; MASK
;2556           CSR1.MASK=17            ; MASK (01003FFF)
;2557           CSR2.MASK=18            ; MASK (7FFE3FFF)
;2558           #FE7FFFFF=19            ; MASK
;2559           UBS.SET=1A              ; CONSTANT (7FF80000) USED BY UBS ADDRESS TEST
;2560           UBS.DATA=1B             ; MASK (7FFF8000) USED AS ERROR MASK FOR UBS
;2561                                   ; DATA TEST
;2562
;2563           ERR.CON.NA=1E           ; CONSTANT OF 800500(H) BITS 23,10,8 SET FOR
;2564                                   ; LOADING CONTROL AND STATUS REG IN INITIAL
;2565                                   ; TESTS
;2566           ERR.CON=1F              ; CONSTANT OF 500(H) BITS 10 & 8 SET FOR
;2567                                   ; LOADING CONTROL AND STATUS REG IN INITIAL
;2568                                   ; TESTS
;2569
;2570           #1=20                   ; PATTERN 00000001 (HEX)
;2571           #2=21                   ; PATTERN 00000002
;2572           #4=22                   ; PATTERN 00000004
;2573           #8=23                   ; PATTERN 00000008
;2574           #10=24                  ; PATTERN 00000010
;2575           #20=25                  ; PATTERN 00000020
```

```
;2576            #40=26                  ; PATTERN 00000040
;2577            #80=27                  ; PATTERN 00000080
;2578            #100=28                 ; PATTERN 00000100
;2579            #200=29                 ; PATTERN 00000200
;2580            #400=2A                 ; PATTERN 00000400
;2581            #800=2B                 ; PATTERN 00000800
;2582            #1000=2C                ; PATTERN 00001000
;2583            #2000=2D                ; PATTERN 00002000
;2584            #4000=2E                ; PATTERN 00004000
;2585            #8000=2F                ; PATTERN 00008000
;2586            #10000=30               ; PATTERN 00010000
;2587            #20000=31               ; PATTERN 00020000
;2588            #40000=32               ; PATTERN 00040000
;2589            #80000=33               ; PATTERN 00080000
;2590            #100000=34              ; PATTERN 00100000
;2591            #200000=35              ; PATTERN 00200000
;2592            #400000=36              ; PATTERN 00400000
;2593            #800000=37              ; PATTERN 00800000
;2594            #1000000=38             ; PATTERN 01000000
;2595            #2000000=39             ; PATTERN 02000000
;2596            #4000000=3A             ; PATTERN 04000000
;2597            #8000000=3B             ; PATTERN 08000000
;2598            #10000000=3C            ; PATTERN 10000000
;2599            #20000000=3D            ; PATTERN 20000000
;2600            #40000000=3E            ; PATTERN 40000000
;2601            #80000000=3F            ; PATTERN 80000000
;2602
;2603            BIT0=20                 ; PATTERN 00000001
;2604            BIT1=21                 ; PATTERN 00000002
;2605            BIT2=22                 ; PATTERN 00000004
;2606            BIT3=23                 ; PATTERN 00000008
;2607            BIT4=24                 ; PATTERN 00000010
;2608            BIT5=25                 ; PATTERN 00000020
;2609            BIT6=26                 ; PATTERN 00000040
;2610            BIT7=27                 ; PATTERN 00000080
;2611            BIT8=28                 ; PATTERN 00000100
;2612            BIT9=29                 ; PATTERN 00000200
;2613            BIT10=2A                ; PATTERN 00000400
;2614            BIT11=2B                ; PATTERN 00000800
;2615            BIT12=2C                ; PATTERN 00001000
;2616            BIT13=2D                ; PATTERN 00002000
;2617            BIT14=2E                ; PATTERN 00004000
;2618            BIT15=2F                ; PATTERN 00008000
;2619            BIT16=30                ; PATTERN 00010000
;2620            BIT17=31                ; PATTERN 00020000
;2621            BIT18=32                ; PATTERN 00040000
;2622            BIT19=33                ; PATTERN 00080000
;2623            BIT20=34                ; PATTERN 00100000
;2624            BIT21=35                ; PATTERN 00200000
;2625            BIT22=36                ; PATTERN 00400000
;2626            BIT23=37                ; PATTERN 00800000
;2627            BIT24=38                ; PATTERN 01000000
;2628            BIT25=39                ; PATTERN 02000000
;2629            BIT26=3A                ; PATTERN 04000000
;2630            BIT27=3B                ; PATTERN 08000000
```

```
;2631            BIT28=3C                ; PATTERN 10000000
;2632            BIT29=3D                ; PATTERN 20000000
;2633            BIT30=3E                ; PATTERN 40000000
;2634            BIT31=3F                ; PATTERN 80000000
;2635
;2636
;2637    ;CSR ADDRESS MNEMONICS
;2638
;2639            CSR0=4E                 ; ALL BITS CLEAR TO ACCESS CSR 0
;2640            CSR1=22                 ; BIT 2 SET TO ACCESS CSR 1
;2641            CSR2=23                 ; BIT 3 SET TO ACCESS CSR 2
;2642
;2643    ;CONTROL AND STATUS WORD BITS
;2644
;2645            PRINT.UBE=26            ; PRINT IF UBE PRESENT OR NOT PRESENT (BIT 6)
;2646                                    ;  (INDICATOR IN LS 7)
;2647            PRINT.R80=27            ; PRINT IF R80 PRESENT OR NOT PRESENT AND ON
;2648                                    ; WHICH DRIVE (BIT 7).  INDICATOR IN LS 7,
;2649                                    ; DRIVE NUMBER IN LS 6.
;2650            ERROR=28                ; TEST ERROR INDICATION (BIT 8)
;2651            SET.PA.ERR=29           ; TELLS CONSOLE TO CREATE A PARITY ERROR AT WCS
;2652                                    ;  ADDRESS POINTED TO BY LS 7 (BIT 9)
;2653            CONSOLE.LOE=2A          ; INDICATES CONSOLE DOES ERROR LOOPING (BIT 10)
;2654                                    ;  (FOR INITIAL TESTS)
;2655            PRINT.MEM.SIZE=2B       ; PRINT MEMORY SIZE IN 256K BLOCKS (BIT 11)
;2656                                    ;  (SIZE IN LS 7)
;2657            PRINT.FPA=2C            ; PRINT IF FPA PRESENT OR NOT PRESENT (BIT 12)
;2658                                    ;  (INDICATOR IN LS 7)
;2659            XFER.DATA=2D            ; INDICATES A DATA TRANSFER TO LS OR MM (BIT 13)
;2660            EOS=2E                  ; END OF SECTION (BIT 14)
;2661            BEGIN.TEST=2F           ; BEGINNING OF TEST (BIT 15)
;2662            INTERUPT.EN=30          ; ENABLE INTERUPT OR SIGNAL SPECIFIED BY BITS
;2663                                    ;  17 THRU 22 AND 28 THRU 30 (BIT 16)
;2664
;2665             CONSOLE.HALT=31        ;   CONSOLE HALT INTERRUPT (BIT 17)
;2666             PWR.FAIL=32            ;   PWR FAIL INT (BIT 18)
;2667             INTERVAL.TIM=33        ;   INTRVL TIM INT (BIT 19)
;2668             CONSOLE.ATTN=34        ;   CONSOLE ATTN INTERRUPT (BIT 20)
;2669             CONSOLE.ACK=35         ;   CONSOLE ACK INTERRUPT (BIT 21)
;2670             DISABLE.MEM.REF=36     ;   DISABLE MEMORY REFERANCES (BIT 22)
;2671             CINIT=3C               ;   UNIBUS INIT SIGNAL TO THE MCT (BIT 28)
;2672             UBS.DCLO=3D            ;   UNIBUS DC LO INDICATION TO THE MCT (BIT 29)
;2673             UBS.BBSY=3E            ;   UNIBUS BUS BUSY INDICATOR TO MCT (BIT 30)
;2674
;2675            NA.EXP.REC=37           ; PRINT N/A UNDER EXPECTED AND RECEIVED (BIT 23)
;2676            OTHER.DATA=38           ; PRINT A VALUE UNDER OTHER DATA (BIT 24)
;2677            CONSOLE.LOOP=39         ; CONSOLE PERFORMS A LOOP ACCORDING TO LOOP
;2678                                    ; COUNT IN LS (BIT 25)
;2679            PRINT.CRD=3A            ; PRINT NUMBER OF SINGLE BIT ERRORS (BIT 26)
;2680            CLR.PA.ERR=3B           ; TELLS CONSOLE TO CLEAR PARITY ERROR AT WCS
;2681                                    ;  ADDRESS POINTED TO BY LS 7 (BIT 27)
;2682            PRINT.IDC=3F            ; PRINT IF IDC PRESENT OR NOT PRESENT (BIT 31)
;2683                                    ;  (INDICATOR IN LS 7)
;2684
;2685    ;MODULE CODES
```

```
;2686
;2687            WCS=20                  ; MODULE CODE FOR WCS BOARD (BIT 0 SET)
;2688            CPU=21                  ; MODULE CODE FOR CPU BOARD (BIT 1 SET)
;2689            MCT=22                  ; MODULE CODE FOR MCT BOARD (BIT 2 SET)
;2690            FPA=23                  ; MODULE CODE FOR FPA BOARD (BIT 3 SET)
;2691            ARRAY=24                ; MODULE CODE FOR ARRAY BOARD (BIT 4 SET)
;2692            IDC=25                  ; MODULE CODE FOR IDC BOARD (BIT 5 SET)
;2693
;2694    ;CSR 1 ERROR BITS
;2695
;2696            VALID.ERR=2E            ; VALID NOT SET IF 1 (BIT 14)
;2697            TB.PAR.ERR=2F           ; TB PARITY ERROR (BIT 15)
;2698            NXM=30                  ; NON-EXISTANT MEMORY ERROR (BIT 16)
;2699            UB.BSY=31               ; UNIBUS BUSY (BIT 17)
;2700            ADP.REG=32              ; UNIBUS ADAPTER REGISTER SELECT (BIT 18)
;2701            WR.ACROSS.PG=33         ; WRITE ACROSS PAGE BOUNDARY ERROR (BIT 19)
;2702            OP.ERR=34               ; OPERATION ERROR (BIT 20)
;2703            TB.MISS=35              ; TB MISS (VIRTUAL ADDRESS TRANSLATION NOT IN
;2704                                    ;   BUFFER) (BIT 21)
;2705            ACCESS.REFUSED=36       ; PROTECTION ERROR ON ACCESS (BIT 22)
;2706            MODIFY.REFUSED=37       ; PROTECTION ERROR ON WRITE (BIT 23)
;2707
;2708            CRD=3E                  ; SINGLE BIT ERROR CORRECTED (BIT 30)
;2709            RDS=3F                  ; UNCORRECTABLE DATA ERROR (BIT 31)
;2710
;2711
;2712    ;CSR 1 CONTROL BITS
;2713
;2714            ECC.DIS=39              ; CSR1 ECC DISABLE BIT (BIT 25)
;2715            DIAG.CHK=3A             ; CSR1 DIAG CHK BIT (BIT 26)
;2716            MME=3B                  ; CSR1 MEMORY MANAGMENT ENABLE BIT (BIT 27)
;2717            INH.CRD=3C              ; CSR1 INHIBIT REPORT CRD BIT (BIT 28)
;2718            TB.PAR.DIAG=3D          ; CSR1 TB PAR DIAG BIT (FORCE TB PARITY ERR)
;2719                                    ;  (BIT 29)
;2720
;2721
;2722    ;UBE CONTROL REGISTER BITS
;2723
;2724      ;CONTROL REG 1
;2725            GO=20                   ; START UBE (BIT 0)
;2726            BR4=21                  ; ISSUE BUS REQUEST 4 (BIT 1)
;2727            BR5=22                  ; ISSUE BUS REQUEST 5 (BIT 2)
;2728            BR6=23                  ; ISSUE BUS REQUEST 6 (BIT 3)
;2729            BR7=24                  ; ISSUE BUS REQUEST 7 (BIT 4)
;2730            NPR=25                  ; ISSUE NPR (BIT 5)
;2731            INT.ODNE=26             ; INTERRUPT ON DONE (WHEN CCOVF) (BIT 6)
;2732            RDY=27                  ; READY BIT, SET WHEN READY TO BEGIN FUNC (BIT 7)
;2733            C00=28                  ; C0 BIT (BIT 8)
;2734            C01=29                  ; C1 BIT (BIT 9)
;2735            FUNA=2A                 ; FUNCTION BIT A (BIT 10
;2736            FUNB=2B                 ; FUNCTION BIT B (BIT 11)
;2737            CC+DAT=2C               ; DATA FROM CYCLE COUNT IF SET, DATA REG IF CLEAR
;2738            INH.DATIP.ROL=2D        ; INHIBIT ROL ON DATIP (BIT 13)
;2739            DATOB.ON.DATIP=2E       ; DO DATOB AFTER DATIP CYCLE (BIT 14)
;2740            ERR=2F                  ; EXERCISOR OR UNIBUS ERROR (BIT 15)
```

```
;2741
;2742      ;CONTROL REG 2
;2743            EXT.MEM0=20             ; EXTENDED MEMORY BIT 0 (BIT 0)
;2744            EXT.MEM1=21             ; EXTENDED MEMORY BIT 1 (BIT 1)
;2745            INH.INC.ADDR&CC=22      ; INHIBIT INC ON CYCLE COUNT AND ADDR REG (BIT 2)
;2746            INH.SACK=23             ; INHIBIT ASSERING BUS SACK ON BUS GRANT (BIT 3)
;2747            PWR.DWN.SEQ=24          ; ASSERT ACLO (BIT 4)
;2748            WNG.GNT.BCK=25          ; NO BUS GRANT RECV TO HIGEST REQUEST (BIT 5)
;2749            MAX.LATE.ERR=26         ; NPR AND BR LATENCY TIME EXCEEDED (BIT 6)
;2750            NO.SACK.ERR=27          ; NO TIMEOUT AT CPU WHEN NO SACK (BIT 7)
;2751            NO.SSYN.ERR=28          ; NO SSYN RECEIVED AFTER ASSERTING MSYN (BIT 8)
;2752            WRG.A.LINES=29          ; ADDRESS ERR ON SENT & RECEIVED ADDRESS (BIT 9)
;2753            NO.GNT+NOT.ONE.GNT=2A   ; NO GRANT OR MORE THAN 1 GRANT AFTER REQ. (BIT 10)
;2754            INTR.SSYN.ERR=2B        ; NO SSYN RECEIVED AFTER BUS INTR (BIT 11)
;2755            ENB.PB=2C               ; ENABLE PARITY BIT B (BIT 12)
;2756            CCOVF=2D                ; CYCLE COUNT OVERFLOW (BIT 13)
;2757            TM.DLY=2E               ; REQUEST BUS EVERY 10 USEC (BIT 14)
;2758
;2759      ;BG6 MNEMONIC
;2760            BG6=23                  ; BIT 3 SET FOR BG 6 ADDRESS
;2761
;2762      ;ERROR AND CONTROL INFORMATION
;2763
;2764            CONTROL.STATUS=40       ; CONTROL AND STATUS WORD
;2765            ERROR.NUMBER=41         ; ERROR NUMBER OF CURRENT TEST
;2766            EXPECTED.DATA=42        ; EXPECTED RESULT OF CURRENT TEST
;2767            RECEIVED.DATA=43        ; ACTUAL RESULT OF CURRENT TEST
;2768            ADDRESS.DATA=44         ; OTHER PERTINENT DATA
;2769            ERROR.MASK=45           ; BITS TO CHECK IN RESULT
;2770            MODULE.NUM=46           ; STORAGE OF MODULE NUMBER (CODED)
;2771            LOOP.CNT=47             ; STORAGE OF LOOP COUNT FOR CONSOLE LOOP
;2772                                    ; CONTROL.
;2773            ERROR.CONTROL=48        ; STORAGE FOR ERROR CONTROL (LOOP ON ERROR ETC.)
;2774             LOE=20                 ;  LOOP ON ERROR IF SET (BIT0)
;2775             NER=21                 ;  NO ERROR REPORT IF SET (BIT1)
;2776             BELL=22                ;  BELL ON ERROR IF SET (BIT2)
;2777             HOE=23                 ;  HALT ON ERROR IF SET (BIT3)
;2778             SER=24                 ;  ENABLE ERROR REPORT ON CRD ERRORS IF SET
;2779             APT.PRESENT=25         ;  APT PRESENT IF SET (BIT5)
;2780             LOOP.COMMAND=26        ;  LOOP COMMAND TYPED IF SET (BIT 6)
;2781             SPECIAL=27             ;  SPECIAL BIT FOR LOOP ON SIZE TESTS (BIT 7)
;2782
;2783            APT.PARAM=49            ;APT RUN TIME PARAMETER REGS (MEM SIZE, HARD-
;2784                                    ; WARE CONFIG, SOFTWARE CONFIG IN BYTES 0, 1
;2785                                    ; AND 2 RESPECTIVELY. BYTE 3 FOR FUTURE USE)
;2786            APT.RESERVED=4A         ;RESERVED FOR FUTURE APT USE
;2787
;2788    ;MISCELLANEOUS CONSTANTS AND STORAGE
;2789
;2790            #AAAAAAAA=4C            ; CONSTANT
;2791            ALTER.ODD=4C            ; CONSTANT
;2792            #55555555=4D            ; CONSTANT
;2793            ALTER.EVEN=4D           ; CONSTANT
;2794            #0=4E                   ; CONSTANT
;2795            ZERO=4E                 ; CONSTANT
```

```
;2796            #FFFFFFFF=4F            ; CONSTANT
;2797            ONES=4F                 ; CONSTANT
;2798            PREVIOUS.ERROR=50       ; ERROR NUMBER OF LAST ERROR
;2799
;2800            DATA.1=51               ; STORAGE FOR FPA DATA
;2801            DATA.2=52               ; STORAGE FOR FPA DATA
;2802            DATA.3=53               ; STORAGE FOR FPA DATA
;2803            FIRST.FLT=54            ; STORAGE FOR FPA DATA
;2804            SEC.FLT=55              ; STORAGE FOR FPA DATA
;2805            THIRD.FLT=56            ; STORAGE FOR FPA DATA
;2806            T57=57                  ; TEMP LOCATION 57
;2807            T58=58                  ; TEMP LOCATION 58
;2808            T59=59                  ; TEMP LOCATION 59
;2809
;2810            BEDB=5A                 ;UBE (BUS EXERCISOR) DATA BUF REG
;2811            BECC=5B                 ;UBE (BUS EXERCISOR) CYCLE COUNT REG
;2812            BEBA=5C                 ;UBE (BUS EXERCISOR) ADDR REG
;2813            BECR1=5D                ;UBE (BUS EXERCISOR) CONTROL REG 1
;2814            CLEAR.ERR.ADDR=5E       ;UBE CLEAR ERROR REG ADDRESS
;2815            BECR2=5F                ;UBE (BUS EXERCISOR) CONTROL REG 2
;2816
;2817            #3(H)=60                ; CONSTANT
;2818            #12(H)=61               ; CONSTANT
;2819            #33333333=62            ; CONSTANT
;2820            #0F0F0F0F=63            ; CONSTANT
;2821            #00FF00FF=64            ; CONSTANT
;2822            #162(H)=65              ; CONSTANT FOR LS TRANSFER POINTER
;2823            BR7.IDENT=66            ; BR7 IDENTIFIER (1010(B) IN BITS 5-2)
;2824            BG7=67                  ; BG7 ADDRESS (BITS 2 AND 3 SET)
;2825
;2826    ;TEMPS FOR CPU TESTS
;2827            L30=30                  ;LS 30 TEMP (RESTORED TO 10000 AFTER USE)
;2828            L31=31                  ;LS 31 TEMP (RESTORE TO 20000 AFTER USE)
;2829            L53=53                  ;LS 53 TEMP
;2830            L73=73                  ;LS 73 TEMP
;2831
;2832    ;REGISTER ADDRESSES
;2833
;2834            CONTROL.OS=78           ; HARDWARE INDEX TO CONTROL WORDS (LS 40-4F)
;2835            SHIFT.OS(3-0)=79        ; 16 LOCATION HARDWARE INDEX TO SHIFT PATTERN
;2836                                    ;  (LS 20-2F)
;2837            SHIFT.OS(4-0)=7B        ; 32 LOCATION HARDWARE INDEX TO SHIFT PATTERN
;2838                                    ;  (LS 20-3F)
;2839            OS=7C                   ; OS REGISTER
;2840            DEC.CON=7D              ; DECIMAL CARRIES
;2841            SIZE=7E                 ; SIZE REGISTER
;2842            MDT= 7E                 ; MEMORY REFERENCE DATA SIZE
;2843            INTERRUPT.VEC=7E        ; HARDWARE INTERRUPT VECTOR
;2844    ;
;2845    ;       THIS WORD IS THE HARDWARE CONDITION CODES.  THE CC'S CAN BE READ
;2846    ;       OR ARE WRITTEN HERE.  NO OTHER BITS IN THE PSL ARE EFFECTED.
;2847    ;
;2848            PSL.CC=7F               ; PSL CONDITION CODES
```

```
;2849   .PAGE
;2850   XD.ADRS/=<16:9>,.VALIDITY=<XD.VAL>
;2851
;2852
;2853           SAV.WR0=1               ; STORAGE FOR WR0
;2854           SAV.WR1=2               ; STORAGE FOR WR1
;2855           SAV.WR2=3               ; STORAGE FOR WR2
;2856           SAV.WR3=4               ; STORAGE FOR WR3
;2857           T5.SUB=5                ; RESERVED FOR COMMON SUBROUTINES
;2858           T6.SUB=6                ; RESERVED FOR COMMON SUBROUTINES
;2859           T7.SUB=7                ; RESERVED FOR COMMON SUBROUTINES
;2860           TRANSFER.POINTER=7      ; POINTER FOR TRANSFER ROUTINE
;2861           MEMORY.SIZE=7           ; STORAGE FOR MEMORY SIZE IN 256KB BLOCKS AFTER
;2862                                   ;  SIZING
;2863           FPA.FOUND=7             ; STORAGE FOR RESULT OF SIZING FOR FPA PRESENT
;2864           UBE.FOUND=7             ; STORAGE FOR RESULT OF SIZING FOR UBE PRESENT
;2865           T8=8                    ; TEMP LOCATION 8
;2866           T9=9                    ; TEMP LOCATION 9
;2867           T10=0A                  ; TEMP LOCATION 10
;2868           T11=0B                  ; TEMP LOCATION 11
;2869           T12=0C                  ; TEMP LOCATION 12
;2870           T13=0D                  ; TEMP LOCATION 13
;2871           T14=0E                  ; TEMP LOCATION 14
;2872           T15=0F                  ; TEMP LOCATION 15
;2873           PC=10                   ; MACRO PROGRAM COUNTER
;2874
;2875           WR.BACKUP=11            ; BACKUP WR FOR MOV MEM.DATA TO WR[]
;2876           MM.LASTADDR+1=12        ; STORAGE OF LAST ADDRESS IN MAIN MEMORY PLUS
;2877                                   ;  1 (FIRST NON-EXISTANT MEMORY ADDRESS)
;2878           #FF=13                  ; MASK
;2879           #FFFF=14                ; MASK
;2880           #FF000000=15            ; MASK
;2881           #FFFFFF00=16            ; MASK
;2882           CSR0.MASK=16            ; MASK
;2883           CSR1.MASK=17            ; MASK (01003FFF)
;2884           CSR2.MASK=18            ; MASK (FFFE3FFF)
;2885           #FE7FFFFF=19            ; MASK
;2886           UBS.SET=1A              ; CONSTANT (7FF80000) USED BY UBS ADDRESS TEST
;2887           UBS.DATA=1B             ; MASK (7FFF8000) USED AS ERROR MASK FOR UBS
;2888                                   ; DATA TEST
;2889
;2890           ERR.CON.NA=1E           ; CONSTANT OF 800500(H) BITS 23,10,8 SET FOR
;2891                                   ; LOADING CONTROL AND STATUS REG IN INITIAL
;2892                                   ; TESTS
;2893           ERR.CON=1F              ; CONSTANT OF 500(H) BITS 10 & 8 SET FOR
;2894                                   ; LOADING CONTROL AND STATUS REG IN INITIAL
;2895                                   ; TESTS
;2896
;2897           #1=20                   ; PATTERN 00000001 (HEX)
;2898           #2=21                   ; PATTERN 00000002
;2899           #4=22                   ; PATTERN 00000004
;2900           #8=23                   ; PATTERN 00000008
;2901           #10=24                  ; PATTERN 00000010
;2902           #20=25                  ; PATTERN 00000020
;2903           #40=26                  ; PATTERN 00000040
```

```
;2904            #80=27                  ; PATTERN 00000080
;2905            #100=28                 ; PATTERN 00000100
;2906            #200=29                 ; PATTERN 00000200
;2907            #400=2A                 ; PATTERN 00000400
;2908            #800=2B                 ; PATTERN 00000800
;2909            #1000=2C                ; PATTERN 00001000
;2910            #2000=2D                ; PATTERN 00002000
;2911            #4000=2E                ; PATTERN 00004000
;2912            #8000=2F                ; PATTERN 00008000
;2913            #10000=30               ; PATTERN 00010000
;2914            #20000=31               ; PATTERN 00020000
;2915            #40000=32               ; PATTERN 00040000
;2916            #80000=33               ; PATTERN 00080000
;2917            #100000=34              ; PATTERN 00100000
;2918            #200000=35              ; PATTERN 00200000
;2919            #400000=36              ; PATTERN 00400000
;2920            #800000=37              ; PATTERN 00800000
;2921            #1000000=38             ; PATTERN 01000000
;2922            #2000000=39             ; PATTERN 02000000
;2923            #4000000=3A             ; PATTERN 04000000
;2924            #8000000=3B             ; PATTERN 08000000
;2925            #10000000=3C            ; PATTERN 10000000
;2926            #20000000=3D            ; PATTERN 20000000
;2927            #40000000=3E            ; PATTERN 40000000
;2928            #80000000=3F            ; PATTERN 80000000
;2929
;2930            BIT0=20                 ; PATTERN 00000001
;2931            BIT1=21                 ; PATTERN 00000002
;2932            BIT2=22                 ; PATTERN 00000004
;2933            BIT3=23                 ; PATTERN 00000008
;2934            BIT4=24                 ; PATTERN 00000010
;2935            BIT5=25                 ; PATTERN 00000020
;2936            BIT6=26                 ; PATTERN 00000040
;2937            BIT7=27                 ; PATTERN 00000080
;2938            BIT8=28                 ; PATTERN 00000100
;2939            BIT9=29                 ; PATTERN 00000200
;2940            BIT10=2A                ; PATTERN 00000400
;2941            BIT11=2B                ; PATTERN 00000800
;2942            BIT12=2C                ; PATTERN 00001000
;2943            BIT13=2D                ; PATTERN 00002000
;2944            BIT14=2E                ; PATTERN 00004000
;2945            BIT15=2F                ; PATTERN 00008000
;2946            BIT16=30                ; PATTERN 00010000
;2947            BIT17=31                ; PATTERN 00020000
;2948            BIT18=32                ; PATTERN 00040000
;2949            BIT19=33                ; PATTERN 00080000
;2950            BIT20=34                ; PATTERN 00100000
;2951            BIT21=35                ; PATTERN 00200000
;2952            BIT22=36                ; PATTERN 00400000
;2953            BIT23=37                ; PATTERN 00800000
;2954            BIT24=38                ; PATTERN 01000000
;2955            BIT25=39                ; PATTERN 02000000
;2956            BIT26=3A                ; PATTERN 04000000
;2957            BIT27=3B                ; PATTERN 08000000
;2958            BIT28=3C                ; PATTERN 10000000
```

```
;2959            BIT29=3D                ; PATTERN 20000000
;2960            BIT30=3E                ; PATTERN 40000000
;2961            BIT31=3F                ; PATTERN 80000000
;2962
;2963
;2964    ;CSR ADDRESS MNEMONICS
;2965
;2966            CSR0=4E                 ; ALL BITS CLEAR TO ACCESS CSR 0
;2967            CSR1=22                 ; BIT 2 SET TO ACCESS CSR 1
;2968            CSR2=23                 ; BIT 3 SET TO ACCESS CSR 2
;2969
;2970
;2971    ;CONTROL AND STATUS WORD BITS
;2972
;2973            PRINT.UBE=26            ; PRINT IF UBE PRESENT OR NOT PRESENT (BIT 6)
;2974                                    ;  (INDICATOR IN LS 7)
;2975            PRINT.R80=27            ; PRINT IF R80 PRESENT OR NOT PRESENT AND ON
;2976                                    ; WHICH DRIVE (BIT 7).  INDICATOR IN LS 7,
;2977                                    ; DRIVE NUMBER IN LS 6.
;2978            ERROR=28                ; TEST ERROR INDICATION (BIT 8)
;2979            SET.PA.ERR=29           ; TELLS CONSOLE TO CREATE A PARITY ERROR AT WCS
;2980                                    ;  ADDRESS POINTED TO BY LS 7 (BIT 9)
;2981            CONSOLE.LOE=2A          ; INDICATES CONSOLE DOES ERROR LOOPING (BIT 10)
;2982                                    ;  (FOR INITIAL TESTS)
;2983            PRINT.MEM.SIZE=2B       ; PRINT MEMORY SIZE IN 256K BLOCKS (BIT 11)
;2984                                    ;  (SIZE IN LS 7)
;2985            PRINT.FPA=2C            ; PRINT IF FPA PRESENT OR NOT PRESENT (BIT 12)
;2986                                    ;  (INDICATOR IN LS 7)
;2987            XFER.DATA=2D            ; INDICATES A DATA TRANSFER TO LS OR MM (BIT 13)
;2988            EOS=2E                  ; END OF SECTION (BIT 14)
;2989            BEGIN.TEST=2F           ; BEGINNING OF TEST (BIT 15)
;2990            INTERUPT.EN=30          ; ENABLE INTERUPT OR SIGNAL SPECIFIED BY BITS
;2991                                    ;  17 THRU 22 AND 28 THRU 30 (BIT 16)
;2992
;2993             CONSOLE.HALT=31        ;   CONSOLE HALT INTERRUPT (BIT 17)
;2994             PWR.FAIL=32            ;   PWR FAIL INT (BIT 18)
;2995             INTERVAL.TIM=33        ;   INTRVL TIM INT (BIT 19)
;2996             CONSOLE.ATTN=34        ;   CONSOLE ATTN INTERRUPT (BIT 20)
;2997             CONSOLE.ACK=35         ;   CONSOLE ACK INTERRUPT (BIT 21
;2998             DISABLE.MEM.REF=36     ;   DISABLE MEMORY REFERANCES (BIT 22)
;2999             CINIT=3C               ;   UNIBUS INIT SIGNAL TO THE MCT (BIT 28)
;3000             UBS.DCLO=3D            ;   UNIBUS DC LO INDICATION TO THE MCT (BIT 29)
;3001             UBS.BBSY=3E            ;   UNIBUS BUS BUSY INDICATOR TO MCT (BIT 30)
;3002
;3003            NA.EXP.REC=37           ; PRINT N/A UNDER EXPECTED AND RECEIVED (BIT 23)
;3004            OTHER.DATA=38           ; PRINT A VALUE UNDER OTHER DATA (BIT 24)
;3005            CONSOLE.LOOP=39         ; CONSOLE PERFORMS A LOOP ACCORDING TO LOOP
;3006                                    ; COUNT IN LS (BIT 25)
;3007            PRINT.CRD=3A            ; PRINT NUMBER OF SINGLE BIT ERRORS (BIT 26)
;3008            CLR.PA.ERR=3B           ; TELLS CONSOLE TO CLEAR PARITY ERROR AT WCS
;3009                                    ;  ADDRESS POINTED TO BY LS 7 (BIT 27)
;3010            PRINT.IDC=3F            ; PRINT IF IDC PRESENT OR NOT PRESENT (BIT 31)
;3011                                    ;  (INDICATOR IN LS 7)
;3012
;3013
```

```
;3014   ;MODULE CODES
;3015
;3016           WCS=20                  ; MODULE CODE FOR WCS BOARD (BIT 0 SET)
;3017           CPU=21                  ; MODULE CODE FOR CPU BOARD (BIT 1 SET)
;3018           MCT=22                  ; MODULE CODE FOR MCT BOARD (BIT 2 SET)
;3019           FPA=23                  ; MODULE CODE FOR FPA BOARD (BIT 3 SET)
;3020           ARRAY=24                ; MODULE CODE FOR ARRAY BOARD (BIT 4 SET)
;3021           IDC=25                  ; MODULE CODE FOR IDC BOARD (BIT 5 SET)
;3022
;3023   ;CSR 1 ERROR BITS
;3024
;3025           VALID.ERR=2E            ; VALID NOT SET IF 1 (BIT 14)
;3026           TB.PAR.ERR=2F           ; TB PARITY ERROR (BIT 15)
;3027           NXM=30                  ; NON-EXISTANT MEMORY ERROR (BIT 16)
;3028           UB.BSY=31               ; UNIBUS BUSY (BIT 17)
;3029           ADP.REG=32              ; UNIBUS ADAPTER REGISTER SELECT (BIT 18)
;3030           WR.ACROSS.PG=33         ; WRITE ACROSS PAGE BOUNDARY ERROR (BIT 19)
;3031           OP.ERR=34               ; OPERATION ERROR (BIT 20)
;3032           TB.MISS=35              ; TB MISS (VIRTUAL ADDRESS TRANSLATION NOT IN
;3033                                   ;   BUFFER) (BIT 21)
;3034           ACCESS.REFUSED=36       ; PROTECTION ERROR ON ACCESS (BIT 22)
;3035           MODIFY.REFUSED=37       ; PROTECTION ERROR ON WRITE (BIT 23)
;3036
;3037           CRD=3E                  ; SINGLE BIT ERROR CORRECTED (BIT 30)
;3038           RDS=3F                  ; UNCORRECTABLE DATA ERROR (BIT 31)
;3039
;3040
;3041   ;CSR 1 CONTROL BITS
;3042
;3043           ECC.DIS=39              ; CSR1 ECC DISABLE BIT (BIT 25)
;3044           DIAG.CHK=3A             ; CSR1 DIAG CHK BIT (BIT 26)
;3045           MME=3B                  ; CSR1 MEMORY MANAGMENT ENABLE BIT (BIT 27)
;3046           INH.CRD=3C              ; CSR1 INHIBIT REPORT CRD BIT (BIT 28)
;3047           TB.PAR.DIAG=3D          ; CSR1 TB PAR DIAG BIT (FORCE TB PARITY ERR)
;3048                                   ;  (BIT 29)
;3049
;3050
;3051   ;UBE CONTROL REGISTER BITS
;3052
;3053     ;CONTROL REG 1
;3054           GO=20                   ; START UBE (BIT 0)
;3055           BR4=21                  ; ISSUE BUS REQUEST 4 (BIT 1)
;3056           BR5=22                  ; ISSUE BUS REQUEST 5 (BIT 2)
;3057           BR6=23                  ; ISSUE BUS REQUEST 6 (BIT 3)
;3058           BR7=24                  ; ISSUE BUS REQUEST 7 (BIT 4)
;3059           NPR=25                  ; ISSUE NPR (BIT 5)
;3060           INT.ODNE=26             ; INTERRUPT ON DONE (WHEN CCOVF) (BIT 6)
;3061           RDY=27                  ; READY BIT, SET WHEN READY TO BEGIN FUNC (BIT 7)
;3062           C00=28                  ; C0 BIT (BIT 8)
;3063           C01=29                  ; C1 BIT (BIT 9)
;3064           FUNA=2A                 ; FUNCTION BIT A (BIT 10
;3065           FUNB=2B                 ; FUNCTION BIT B (BIT 11)
;3066           CC+DAT=2C               ; DATA FROM CYCLE COUNT IF SET, DATA REG IF CLEAR
;3067           INH.DATIP.ROL=2D        ; INHIBIT ROL ON DATIP (BIT 13)
;3068           DATOB.ON.DATIP=2E       ; DO DATOB AFTER DATIP CYCLE (BIT 14)
```

```
;3069            ERR=2F                  ; EXERCISOR OR UNIBUS ERROR (BIT 15)
;3070
;3071      ;CONTROL REG 2
;3072            EXT.MEM0=20             ; EXTENDED MEMORY BIT 0 (BIT 0)
;3073            EXT.MEM1=21             ; EXTENDED MEMORY BIT 1 (BIT 1)
;3074            INH.INC.ADDR&CC=22      ; INHIBIT INC ON CYCLE COUNT AND ADDR REG (BIT 2)
;3075            INH.SACK=23             ; INHIBIT ASSERING BUS SACK ON BUS GRANT (BIT 3)
;3076            PWR.DWN.SEQ=24          ; ASSERT ACLO (BIT 4)
;3077            WNG.GNT.BCK=25          ; NO BUS GRANT RECV TO HIGEST REQUEST (BIT 5)
;3078            MAX.LATE.ERR=26         ; NPR AND BR LATENCY TIME EXCEEDED (BIT 6)
;3079            NO.SACK.ERR=27          ; NO TIMEOUT AT CPU WHEN NO SACK (BIT 7)
;3080            NO.SSYN.ERR=28          ; NO SSYN RECEIVED AFTER ASSERTING MSYN (BIT 8)
;3081            WRG.A.LINES=29          ; ADDRESS ERR ON SENT & RECEIVED ADDRESS (BIT 9)
;3082            NO.GNT+NOT.ONE.GNT=2A   ; NO GRANT OR MORE THAN 1 GRANT AFTER REQ. (BIT 10)
;3083            INTR.SSYN.ERR=2B        ; NO SSYN RECEIVED AFTER BUS INTR (BIT 11)
;3084            ENB.PB=2C               ; ENABLE PARITY BIT B (BIT 12)
;3085            CCOVF=2D                ; CYCLE COUNT OVERFLOW (BIT 13)
;3086            TM.DLY=2E               ; REQUEST BUS EVERY 10 USEC (BIT 14)
;3087
;3088     ;BG6 MNEMONIC
;3089            BG6=23                  ; BIT 3 SET FOR BG 6 ADDRESS
;3090
;3091     ;ERROR AND CONTROL INFORMATION
;3092
;3093            CONTROL.STATUS=40       ; CONTROL AND STATUS WORD
;3094            ERROR.NUMBER=41         ; ERROR NUMBER OF CURRENT TEST
;3095            EXPECTED.DATA=42        ; EXPECTED RESULT OF CURRENT TEST
;3096            RECEIVED.DATA=43        ; ACTUAL RESULT OF CURRENT TEST
;3097            ADDRESS.DATA=44         ; OTHER PERTINENT DATA
;3098            ERROR.MASK=45           ; BITS TO CHECK IN RESULT
;3099            MODULE.NUM=46           ; STORAGE OF MODULE NUMBER (CODED)
;3100            LOOP.CNT=47             ; STORAGE OF LOOP COUNT FOR CONSOLE LOOP
;3101                                    ; CONTROL.
;3102            ERROR.CONTROL=48        ; STORAGE FOR ERROR CONTROL (LOOP ON ERROR ETC.)
;3103             LOE=20                 ;  LOOP ON ERROR IF SET (BIT0)
;3104             NER=21                 ;  NO ERROR REPORT IF SET (BIT1)
;3105             BELL=22                ;  BELL ON ERROR IF SET (BIT2)
;3106             HOE=23                 ;  HALT ON ERROR IF SET (BIT3)
;3107             SER=24                 ;  ENABLE ERROR REPORT ON CRD ERRORS IF SET
;3108             APT.PRESENT=25         ;  APT PRESENT IF SET (BIT5)
;3109             LOOP.COMMAND=26        ;  LOOP COMMAND TYPED IF SET (BIT 6)
;3110             SPECIAL=27             ;  SPECIAL BIT FOR LOOP ON SIZE TESTS (BIT 7)
;3111
;3112            APT.PARAM=49            ;APT RUN TIME PARAMETER REGS (MEM SIZE, HARD-
;3113                                    ; WARE CONFIG, SOFTWARE CONFIG IN BYTES 0, 1
;3114                                    ; AND 2 RESPECTIVELY. BYTE 3 FOR FUTURE USE)
;3115            APT.RESERVED=4A         ;RESERVED FOR FUTURE APT USE
;3116
;3117     ;MISCELLANEOUS CONSTANTS AND STORAGE
;3118
;3119            #AAAAAAAA=4C            ; CONSTANT
;3120            ALTER.ODD=4C            ; CONSTANT
;3121            #55555555=4D            ; CONSTANT
;3122            ALTER.EVEN=4D           ; CONSTANT
;3123            #0=4E                   ; CONSTANT
```

```
;3124            ZERO=4E                 ; CONSTANT
;3125            #FFFFFFFF=4F            ; CONSTANT
;3126            ONES=4F                 ; CONSTANT
;3127            PREVIOUS.ERROR=50       ; ERROR NUMBER OF LAST ERROR
;3128
;3129            DATA.1=51               ; STORAGE FOR FPA DATA
;3130            DATA.2=52               ; STORAGE FOR FPA DATA
;3131            DATA.3=53               ; STORAGE FOR FPA DATA
;3132            FIRST.FLT=54            ; STORAGE FOR FPA DATA
;3133            SEC.FLT=55              ; STORAGE FOR FPA DATA
;3134            THIRD.FLT=56            ; STORAGE FOR FPA DATA
;3135            T57=57                  ; TEMP LOCATION 57
;3136            T58=58                  ; TEMP LOCATION 58
;3137            T59=59                  ; TEMP LOCATION 59
;3138
;3139            BEDB=5A                 ;UBE (BUS EXERCISOR) DATA BUF REG
;3140            BECC=5B                 ;UBE (BUS EXERCISOR) CYCLE COUNT REG
;3141            BEBA=5C                 ;UBE (BUS EXERCISOR) ADDR REG
;3142            BECR1=5D                ;UBE (BUS EXERCISOR) CONTROL REG 1
;3143            CLEAR.ERR.ADDR=5E       ;UBE CLEAR ERROR REG ADDRESS
;3144            BECR2=5F                ;UBE (BUS EXERCISOR) CONTROL REG 2
;3145
;3146            #3(H)=60                ; CONSTANT
;3147            #12(H)=61               ; CONSTANT
;3148            #33333333=62            ; CONSTANT
;3149            #0F0F0F0F=63            ; CONSTANT
;3150            #00FF00FF=64            ; CONSTANT
;3151            #162(H)=65              ; CONSTANT FOR LS TRANSFER POINTER
;3152            BR7.IDENT=66            ; BR7 IDENTIFIER (1010(B) IN BITS 5-2)
;3153            BG7-67                  ; BG7 ADDRESS (BITS 2 AND 3 SET)
;3154
;3155    ;TEMPS FOR CPU TESTS
;3156            L30=30                  ;LS 30 TEMP (RESTORED TO 10000 AFTER USE)
;3157            L31=31                  ;LS 31 TEMP (RESTORE TO 20000 AFTER USE)
;3158            L53=53                  ;LS 53 TEMP
;3159            L73=73                  ;LS 73 TEMP
;3160
;3161    ;REGISTER ADDRESSES
;3162
;3163            CONTROL.OS=78           ; HARDWARE INDEX TO CONTROL WORDS (LS 40-4F)
;3164            SHIFT.OS(3-0)=79        ; 16 LOCATION HARDWARE INDEX TO SHIFT PATTERN
;3165                                    ;  (LS 20-2F)
;3166            SHIFT.OS(4-0)=7B        ; 32 LOCATION HARDWARE INDEX TO SHIFT PATTERN
;3167                                    ;  (LS 20-3F)
;3168            OS=7C                   ; OS REGISTER
;3169            DEC.CON=7D              ; DECIMAL CARRIES
;3170            SIZE=7E                 ; SIZE REGISTER
;3171            MDT= 7E                 ; MEMORY REFERENCE DATA SIZE
;3172            INTERRUPT.VEC=7E        ; HARDWARE INTERRUPT VECTOR
;3173    ;
;3174    ;       THIS WORD IS THE HARDWARE CONDITION CODES.  THE CC'S CAN BE READ
;3175    ;       OR ARE WRITTEN HERE.  NO OTHER BITS IN THE PSL ARE EFFECTED.
;3176    ;
;3177            PSL.CC=7F               ; PSL CONDITION CODES
;3178
```

```
;3179
;3180   .TOC    "COMMON LS ASSIGNMENTS (TO REGISTERS)"
;3181   ;
;3182   ;UPPER HALF OF LS
;3183   ;
;3184           XGPR.OS=0F8             ; HARDWARE INDEX TO XGPRS
;3185           USER.INDEX(3-0)=0F9     ; 16 LOCATION HARDWARE INDEX TO DIAGNOSTIC DATA
;3186                                   ;   AREA (LS LOCATIONS 0A0 THROUGH 0AF)
;3187           USER.INDEX(4-0)=0FB     ; 32 LOCATION HARDWARE INDEX TO DIAGNOSTIC DATA
;3188                                   ;   AREA (LS LOCATIONS 0A0 THROUGH 0BF)
;3189           CCR=0FC                 ; CONSOLE READ REGISTER
;3190           TIMER=0FD               ; INTERVAL TIMER VALUE.
;3191           ALU.CC=0FE              ; ALU CONDITION CODES
;3192   ;
;3193   ;       THIS LOCATION IS A WRITE ONLY REGISTER.  THE FOLLOWING BITS IN THE
;3194   ;       PSL ARE AFFECTED:
;3195   ;
;3196   ;       COMPATIBILITY MODE - BIT<31>
;3197   ;       CURRENT MODE - BITS<25:24>
;3198   ;       INTERRUPT PRIORITY LEVEL (IPL) - BITS<20:16>
;3199   ;       TRACE TRAP ENABLE (REALLY T OR TP) - BIT<4>
;3200   ;       NEGATIVE CONDITION CODE - BIT<3>
;3201   ;       ZERO CONDITION CODE - BIT<2>
;3202   ;       OVERFLOW CONDITION CODE - BIT<1>
;3203   ;       CARRY CONDITION CODE - BIT<0>
;3204   ;
;3205           PSL.HW=0FF              ; HARDWARE PSL BITS
;3206   .TOC    "PROGRAM SPECIFIC LS ASSIGNMENTS (LS 80-F7)"
;3207   ;
;3208   ; THESE ADDRESSES ARE ONLY ACCESSIBLE WITH THE MOV MICRO INSTRUCTION.
;3209   ; STARTING AT ADDRESS 80(H).  THEY ARE NOT USED BY ALL SOFTWARE MODULES
;3210   ; BUT ARE SPECIFIC TO THIS ONE.
;3211   ;
;3212
;3213           CKBTS.0111100=80        ;COMPLEMENT OF EXP CKBTS FOR CKBT GEN TEST
;3214           CKBTS.1000011=81        ;COMPLEMENT OF EXP CKBTS FOR CKBT GEN TEST
;3215           CKBTS.1100100=82        ;COMPLEMENT OF EXP CKBTS FOR CKBT GEN TEST
;3216           CKBTS.0100101=83        ;COMPLEMENT OF EXP CKBTS FOR CKBT GEN TEST
;3217           CKBTS.0100110=84        ;COMPLEMENT OF EXP CKBTS FOR CKBT GEN TEST
;3218           CKBTS.0100000=85        ;COMPLEMENT OF EXP CKBTS FOR CKBT GEN TEST
;3219           CKBTS.1010100=86        ;COMPLEMENT OF EXP CKBTS FOR CKBT GEN TEST
;3220           CKBTS.1111101=87        ;COMPLEMENT OF EXP CKBTS FOR ECC LATCH TEST
;3221
;3222           IDENT.MASK=88           ;MASK FOR BR IDENTIFIER (FFFFFFC3)
;3223
;3224           #FFFFFF80=89            ;ERROR MASK FOR BITS 6-0 (CSR 0 MASK)
;3225
;3226           #30000=9A               ;DATA PATTERN FOR DOUBLE BIT ERROR
;3227           #7FFF8000=9B            ;MASK FOR TAG FIELD TEST (BITS 15-30 SET)
;3228           #540000=9C              ;ADDRESS FOR USE IN NXM TEST
;3229           #F00000=9D              ;ADDRESS FOR USE IN NXM TEST
;3230           #FC0000=9E              ;ADDRESS FOR USE IN NXM TEST
;3231
;3232           #3F003FFF=9F            ;MASK FOR CSR 1 ERROR BITS
;3233
```

```
;3234          ACCESS.RCHK.KERNAL=0A0  ;EXPECTED DATA FOR PROTECTION TEST
;3235          MODIFY.RCHK.KERNEL=0A1  ;EXPECTED DATA FOR PROTECTION TEST
;3236
;3237          ACCESS.RCHK.EXEC=0A2    ;EXPECTED DATA FOR PROTECTION TEST
;3238          MODIFY.RCHK.EXEC=0A3    ;EXPECTED DATA FOR PROTECTION TEST
;3239
;3240          ACCESS.RCHK.SUPER=0A4   ;EXPECTED DATA FOR PROTECTION TEST
;3241          MODIFY.RCHK.SUPER=0A5   ;EXPECTED DATA FOR PROTECTION TEST
;3242
;3243          ACCESS.RCHK.USER=0A6    ;EXPECTED DATA FOR PROTECTION TEST
;3244          MODIFY.RCHK.USER=0A7    ;EXPECTED DATA FOR PROTECTION TEST
;3245
;3246
;3247          ACCESS.WCHK.KERNAL=0A8  ;EXPECTED DATA FOR PROTECTION TEST
;3248          MODIFY.WCHK.KERNEL=0A9  ;EXPECTED DATA FOR PROTECTION TEST
;3249
;3250          ACCESS.WCHK.EXEC=0AA    ;EXPECTED DATA FOR PROTECTION TEST
;3251          MODIFY.WCHK.EXEC=0AB    ;EXPECTED DATA FOR PROTECTION TEST
;3252
;3253          ACCESS.WCHK.SUPER=0AC   ;EXPECTED DATA FOR PROTECTION TEST
;3254          MODIFY.WCHK.SUPER=0AD   ;EXPECTED DATA FOR PROTECTION TEST
;3255
;3256          ACCESS.WCHK.USER=0AE    ;EXPECTED DATA FOR PROTECTION TEST
;3257          MODIFY.WCHK.USER=0AF    ;EXPECTED DATA FOR PROTECTION TEST
;3258
```

```
;3259   .page 'Microinstruction Formats''
;3260   ;
;3261   ;       The following is taken directly from the NEBULA Hardware
;3262   ;       Specification.
;3263   ;
;3264   ;                       MICROINSTRUCTION FORMATS
;3265   ;
;3266   ;
;3267   ;
;3268   ;
;3269   ;                   22 21       16 15          9 8 7 6 5 4       0
;3270   ;                   +-----+-------+-------+-------+-------+-------+
;3271   ;1. BASIC           |1|     DP    |    D ADRS     | B |CC |  SCTL   |
;3272   ;                   +-+-----------+---------------+---+---+---------+
;3273   ;
;3274   ;                   22  20 19 17 16            9 8 7 6 5 4       0
;3275   ;                   +-----+-------+-------+-------+-------+-------+
;3276   ;2. MOVE            |0 1 1| MDP   |    XD ADRS    | B |CC |  SCTL   |
;3277   ;                   +-----+-------+---------------+---+---+---------+
;3278   ;
;3279   ;                   22  20 19       14 13 11   9 8 7 6 5 4       0
;3280   ;                   +-----+-------+-------+-------+-------+-------+
;3281   ;3. EXTENDED        |0 1 0|    XDP    | SI  | A | B |CC |  SCTL   |
;3282   ;                   +-----+-----------+-----+---+---+---+---------+
;3283   ;
;3284   ;                   22    19 18 16 15          9 8 7 6 5 4       0
;3285   ;                   +-----+-------+-------+-------+-------+-------+
;3286   ;4. MEM.REQ         |0 0 1 1| MF1   |    D ADRS     |MF2|DT |  SCTL   |
;3287   ;                   +-------+-------+---------------+---+---+---------+
;3288   ;
;3289   ;                   22    19 18 16 15   12 11  9 8 7 6 5 4       0
;3290   ;                   +-----+-------+-------+-------+-------+-------+
;3291   ;5. MISC            |0 0 1 0|P|0 0|   M1  | M2  |0|R|0 0|  SCTL   |
;3292   ;                   +-------+-+---+-------+-----+-+-+---+---------+
;3293   ;
;3294   ;                   22    19 18 17                      4 3     0
;3295   ;                   +-----+-------+-------+-------+-------+-------+
;3296   ;6. JUMP            |0 0 0 1|OS|    JUMP ADDRESS               | JCTL  |
;3297   ;                   +-------+--+-------------------------------+-------+
;3298   ;
;3299   ;                   22    19 18 17      12 11  9 8 7 6 5 4 3     0
;3300   ;                   +-----+-------+-------+-------+-------+-------+
;3301   ;7. DECODE          |0 0 0 0| | DEC.ADRS  |IFUNC| | | | | | JCTL  |
;3302   ;                   +-------+-+-----------+-----+-+-+-+-+-+-------+
;3303   ;                             |                   | | | | | |
;3304   ;                   BACK UP PC          IB REQ----+ | | | +--OPC/SPEC
;3305   ;                                                   | | |
;3306   ;                               SEL CM HI IR BYTE---+ | +--LOAD RDEST
;3307   ;                                                     |
;3308   ;                                           LOAD OS---+
```

```
;3309   .page
;3310   ;
;3311   ;
;3312   ;1. 'BASIC'' Microinstruction
;3313   ;
;3314   ;    CSR<22> = 1    (OPCODE)
;3315   ;
;3316   ;        This opcode bit causes LS Adrs <7> to be cleared.
;3317   ;
;3318   ;    CSR<21:16>  (Data Path Control)
;3319   ;
;3320   ;        This field controls the data path. It controls the 2901 ALU,
;3321   ;        the ALU data source, the data destination in the 2901 array,
;3322   ;        and the write to the Local Store.
;3323   ;
;3324   ;    CSR<15:9>   (D Adrs <6:0>)
;3325   ;
;3326   ;        This field selects which of the 128 accessable Local Store
;3327   ;        locations to use.
;3328   ;
;3329   ;    CSR<8:7>    (B Adrs)
;3330   ;
;3331   ;        This field selects one of the four Working Registers.
;3332   ;
;3333   ;    CSR<6:5>    (Condition Codes)
;3334   ;
;3335   ;        This field specifies the data type and condition code control.
;3336   ;
;3337   ;    CSR<4:0>    (SCTL)
;3338   ;
;3339   ;        This field specifies the skip condition/micro sequencer
;3340   ;        function.
```

```
;3341   .page
;3342   ;
;3343   ;
;3344   ;2. "MOVE" Microinstruction
;3345   ;
;3346   ;     CSR<22:20> = 011 (OPCODE)
;3347   ;
;3348   ;        This combination of opcode bits allows CSR<16> to control
;3349   ;        LS Adrs <7>. This permits access of LS locations 80 - FF.
;3350   ;
;3351   ;     CSR<19:17> (Data Path Control)
;3352   ;
;3353   ;        This field is decoded to produce some data path control
;3354   ;        information.
;3355   ;
;3356   ;          0 LS[XD] <- WR[B], WR[B] <- MEM DATA* 4 LS[XD] <- MEM DATA*
;3357   ;          1 MEMORY <- LS [XD]*                  5 LS[XD] <- ACC/PORT
;3358   ;          2 Q <- XWR[B]                         6 LS[XD] <- WR[B] + OS
;3359   ;          3 WR[B] <- LS [XD]                    7 LS[XD] <- WR[B]
;3360   ;
;3361   ;     CSR<16:9>  (XD<7:0>)
;3362   ;
;3363   ;        This field specifies which of the 256 Local Store locations to
;3364   ;        use.
;3365   ;
;3366   ;     CSR<8:7>   (B Adrs)
;3367   ;
;3368   ;        This field specifies which of the four Working Registers to
;3369   ;        use. For MDP = 2, this field selects either the Backup PC or
;3370   ;        the VAR.
;3371   ;
;3372   ;     CSR<6:5>   (CC)
;3373   ;
;3374   ;        This field specifies the data type and condition code control.
;3375   ;
;3376   ;     CSR<4:0>   (SCTL)
;3377   ;
;3378   ;        This field specifies the Skip control. See Table 3.
;3379   ;
;3380   ;
;3381   ;* For information on which Working Registers and Local Store locations
;3382   ;  may  be  used for Memory Addresses and data source/destinations, see
;3383   ;  the Memory Management documentation.  Note that  these  restrictions
;3384   ;  are due to microcoding conventions and not hardware.
```

```
;3385    .page
;3386    ;
;3387    ;
;3388    ;3. "EXTENDED" Microinstruction
;3389    ;
;3390    ;       CSR<22:20> = 010    (OPCODE)
;3391    ;
;3392    ;           This opcode causes a function internal to the 2901 array; that
;3393    ;           is, an operation involving only Working Registers and the Q-
;3394    ;           Register.
;3395    ;
;3396    ;       CSR<19:14>  (Data Path Control)
;3397    ;
;3398    ;           This field is decoded to supply the 2901 ALU function code,
;3399    ;           the ALU Source control, the destination code, and the ALU
;3400    ;           Carry-In.
;3401    ;
;3402    ;       CSR<13:11>  (Shift Input)
;3403    ;
;3404    ;           This field selects the data to be shifted into a register,
;3405    ;           when a Shift function is being done.
;3406    ;
;3407    ;       CSR<10:9>   (A Adrs)
;3408    ;
;3409    ;           This field selects which Working Register to use as a source
;3410    ;           of data, when RAM A is selected to the ALU.
;3411    ;
;3412    ;       CSR<8:7>    (B Adrs)
;3413    ;
;3414    ;           This field selects which Working Register to use as a source
;3415    ;           (when RAM B is selected to the ALU) and/or destination (when
;3416    ;           the RAM is written) of data.
;3417    ;
;3418    ;       CSR<6:5>    (CC)
;3419    ;
;3420    ;           This field specifies the data type and condition code control.
;3421    ;
;3422    ;       CSR<4:0>    (SCTL)
;3423    ;
;3424    ;           This field specifies the skip condition/micro sequencer
;3425    ;           function.
```

```
;3426    .page
;3427    ;
;3428    ;
;3429    ;4. 'MEM.REQ'' Microinstruction
;3430    ;
;3431    ;    CSR<22:19> = 0011    (OPCODE)
;3432    ;
;3433    ;        This opcode causes a Memory Request to be started. The Local
;3434    ;        Store is output on on the D-Bus, to be used as the virtual
;3435    ;        address. Working Register 5 is written with this address.
;3436    ;
;3437    ;    CSR<18:16>, <8:7>   (Memory Function)
;3438    ;
;3439    ;        This field specifies to the Memory Controller the type of
;3440    ;        request: physical, virtual, type of access, etc. For the
;3441    ;        list of functions and the codes, see the NEBULA Memery Spec.
;3442    ;
;3443    ;    CSR<15:9>   (D Adrs <6:0>)
;3444    ;
;3445    ;        This field selects which of the 128 accessable Local Store
;3446    ;        locations to use as the virtual address.
;3447    ;
;3448    ;    CSR<6:5>    (Data Type)
;3449    ;
;3450    ;        This field specifies the data type of the request.
;3451    ;
;3452    ;                00:=BYTE
;3453    ;                01:=WORD
;3454    ;                10:=SIZE Reg
;3455    ;                11:=LONG
;3456    ;
;3457    ;    CSR<4:0>    (SCTL)
;3458    ;
;3459    ;        This field specifies the skip condition/micro sequencer
;3460    ;        function.
```

```
;3461   .page
;3462   ;
;3463   ;
;3464   ;5. 'MISC" Microinstruction
;3465   ;
;3466   ;    CSR<22:19> = 0010 (OPCODE)
;3467   ;
;3468   ;        This combination of opcode bits causes the data path to no-op,
;3469   ;        and enables the MISC decoders.
;3470   ;
;3471   ;    CSR<18>              (Port/Misc Select)
;3472   ;
;3473   ;        This bit defines the instruction to be either a Misc or a Port
;3474   ;        instruction.   For CSR<18> = 1, CSR<17:10> is the Command Byte
;3475   ;        to the Port, CSR<9:5> must be zero and CSR<4:0>  is  the  SCTL
;3476   ;        field.  For CSR<18> = 0, it is a Misc instruction and the rest
;3477   ;        of the word is defined as follows.
;3478   ;
;3479   ;    CSR<17:16>           (Not Used)
;3480   ;
;3481   ;    CSR<15:12>  (Misc Function 1)
;3482   ;
;3483   ;        These four bits are decoded to do the following functions:
;3484   ;
;3485   ;               0 = CLEAR State Reg<1:0>
;3486   ;               1 = CLEAR State Reg<1>, SET State Reg<0>
;3487   ;               2 = SET State Reg<1>, CLEAR State Reg<0>
;3488   ;               3 = CLEAR State Reg<0>
;3489   ;               4 = CLEAR State Reg<1>
;3490   ;               5 = SET State Reg<0>
;3491   ;               6 = SET State Reg<1>
;3492   ;               7 = SET REGISTER BACKUP MASK FLAG
;3493   ;               8 = SET ACC I/O MODE
;3494   ;               9 = SET PORT I/O MODE
;3495   ;               A = CLEAR WCS HI ADRS
;3496   ;               B = SET WCS HI ADRS
;3497   ;               C = CLEAR CPU ATTN AND CPU ACK
;3498   ;               D = SET CPU ACK
;3499   ;               E = SET CPU ATTN
;3500   ;               F = NOP
;3501   ;
;3502   ;    CSR<11:9>   (Misc Function 2)
;3503   ;
;3504   ;        This field is decoded to do the following functions:
;3505   ;
;3506   ;               0 = NOP
;3507   ;               1 = Mask IRD Interrupt Detection during Next State
;3508   ;               2 = Assert CPU Data Available and pass WR to ACC/Port
;3509   ;               3 = Trap ACC
;3510   ;               4 = Read ACC uPC
;3511   ;               5 = Assert XFER GRANT to Port
;3512   ;               6 = Mask Halt and T-Bit Traps
;3513   ;               7 = Write WR to Console Read Register
```

```
;3514    .page
;3515    ;
;3516    ;    CSR<8>      (MUST BE ZERO)
;3517    ;
;3518    ;    CSR<7>      (WR Address)
;3519    ;
;3520    ;
;3521    ;        This bit selects WR[0] or WR[1] to be put on the Y-Bus.
;3522    ;
;3523    ;    CSR<6:5>            (Not Used)
;3524    ;
;3525    ;    CSR<4:0>    (SCTL)
;3526    ;
;3527    ;        This  field  specifies  the  skip  condition/micro   sequencer
;3528    ;        function.
;3529    ;
```

```
;3530   .page
;3531   ;
;3532   ;
;3533   ;6. "JUMP" Microinstruction
;3534   ;
;3535   ;    CSR<22:19> = 0001    (OPCODE)
;3536   ;
;3537   ;        This opcode causes the data path to no-op, and the micro
;3538   ;        sequencer to execute a JUMP.
;3539   ;
;3540   ;    CSR<18>     (Select OS)
;3541   ;
;3542   ;        This bit, when asserted, causes OS<4:0> to be OR'ed with the
;3543   ;        five low bits of the of the jump microaddress.
;3544   ;
;3545   ;    CSR<17:4>   (Jump Microaddress)
;3546   ;
;3547   ;        If Select OS is CLEAR, this field specifies the fourteen-bit
;3548   ;        jump micro address. If Select OS is SET, CSR<17:9> is Jump
;3549   ;        Adrs<13:5> and OS<4:0> OR'ed with CSR<8:4> is Jump Adrs<4:0>.
;3550   ;
;3551   ;    CSR<3:0>    (JCTL)
;3552   ;
;3553   ;        This field specifies the jump condition/micro sequencer
;3554   ;        function.
```

```
;3555    .page
;3556    ;
;3557    ;
;3558    ;7. "DECODE" Microinstruction
;3559    ;
;3560    ;       A.  DESCRIPTION
;3561    ;
;3562    ;       CSR<22:19> = 0000 (OPCODE)
;3563    ;
;3564    ;           This combination of opcode bits causes the Local Store to
;3565    ;           access location 10 (the PC.) The Local Store drives the D-Bus.
;3566    ;           The 2901 is set up to input the data on the D-Bus, increment
;3567    ;           it, and output to the Y-Bus.  Thus, the Y-Bus contains the PC
;3568    ;           + 1.
;3569    ;
;3570    ;       CSR<18>     (BPC [asserted low])
;3571    ;
;3572    ;           If this bit is CLEAR, the incremented PC is written into the
;3573    ;           2901 RAM.  If it is SET, the RAM is not written.
;3574    ;
;3575    ;       CSR<17:12>  (DEC.ADRS <13:8>)
;3576    ;
;3577    ;           This data is taken to be the upper six bits of the next
;3578    ;           microaddress.  The low eight bits are supplied by the IRD ROM.
;3579    ;           The actual JUMP/JSR/NO-OP is determined by the JCTL field.
;3580    ;
;3581    ;       CSR<11:9>   (IFUNC)
;3582    ;
;3583    ;           This is the IFUNC field.  It defines which fork is being
;3584    ;           taken.
;3585    ;
;3586    ;       CSR<8>      (IB.REQ)
;3587    ;
;3588    ;           This is the IB.REQ bit.  When it is CLEAR, the LS is not
;3589    ;           written, no interaction with memory is initiated and the
;3590    ;           instructin buffer is not affected.
;3591    ;
;3592    ;           When it is SET:
;3593    ;
;3594    ;           The incremented PC (valid on the Y-Bus by the end of the
;3595    ;           cycle) is re-written to Local Store location 0.  This
;3596    ;           completes the PC increment function.
;3597    ;
;3598    ;           A Conditional Memory Request is executed.  If the two low bits
;3599    ;           of the PC are not 11, the request is not initiated.  If the
;3600    ;           two low bits are set, an IB PREFETCH is done.  The
;3601    ;           (unincremented) PC is output to the memory and the CPU grant
;3602    ;           (CPUG) signal is examined.  If CPUG is low by T120, the CPU
;3603    ;           will stall until the CPUG signal becomes true.  After the
;3604    ;           request is completed, the next state entered is dependent upon
;3605    ;           the JCTL field.
```

```
;3606    .page
;3607    ;
;3608    ;    CSR<7>      (SEL CM HI IR BYTE)
;3609    ;
;3610    ;        This bit enables the upper byte of the Compatibility Mode
;3611    ;        instruction in the Prefetch register to drive the IB-Bus. It
;3612    ;        is used only while in Compatibility Mode, when IRD is being
;3613    ;        done. It must never be asserted in VAX Mode.
;3614    ;
;3615    ;
;3616    ;    CSR<6>      (LD.OS)
;3617    ;
;3618    ;        If this bit is SET, the eight-bit data on the IB Bus (the
;3619    ;        output of the instruction buffer) is loaded into the OS
;3620    ;        register. This load is dependent upon Compatibility Mode and
;3621    ;        LD R Dest (If CM.AND.LD R Dest, OS<3> <- 0.) If the bit is
;3622    ;        CLEAR, OS is not affected.
;3623    ;
;3624    ;    CSR<5>      (LD R DEST)
;3625    ;
;3626    ;        If this bit is SET, the R Dest Skip bit will be SET for
;3627    ;        Specifier Mode equal to 5 and CLEARED for Mode not equal to 5,
;3628    ;        in VAX mode, or mode 0 while in Compatibility Mode. If this
;3629    ;        bit is CLEAR, the R Dest bit will be unaffected.
;3630    ;
;3631    ;    CSR<4>              (OPC/SPEC)
;3632    ;
;3633    ;        This bit (effectively another IFUNC bit) defines the data to
;3634    ;        be decoded; that is, if it is an Opcode or a Specifier. In
;3635    ;        hardware, it selects which ROM is to drive the micro address
;3636    ;        bus.
;3637    ;
;3638    ;    CSR<3:0>    (JCTL)
;3639    ;
;3640    ;        These four bits are decoded to make one of 16 functions.  See
;3641    ;        Table 3.
```

```
;3642   .page "Tables"
;3643   ;
;3644   ;                                   TABLES
;3645   ;
;3646   ;
;3647   ;
;3648   ;                       Table 1.  2901 Control Decoding
;3649   ;
;3650   ;
;3651   ;       The 2901 control decoder examines CSR<22:14>. This nine-bit
;3652   ;       field  of  the microword changes in meaning for the different
;3653   ;       cases of micro opcode.
;3654   ;
;3655   ;       CSR      22  21  20  19  18  17  16  15  14
;3656   ;                --  --  --  --  --  --  --  --  --
;3657   ;1. BASIC         1  [Six bit function code]  X   X
;3658   ;2. MOVE          0   1   1  [function]   X   X   X
;3659   ;3. EXTENDED      0   1   0  [Six bit function code]
;3660   ;4. MEM.REQ       0   0   1   1   X   X   X   X   X       WR[5] <- D
;3661   ;5. MISC          0   0   1   0   X   X   X   X   X       Y-Bus <- WR
;3662   ;6. JUMP          0   0   0   1   X   X   X   X   X       NO-OP
;3663   ;7. DECODE        0   0   0   0  BPC* X   X   X   X
;3664   ;
;3665   ;  If BPC = 0, Write WR; else, No-op
;3666   ;
;3667   ;     This decoder is implemented using a ROM and a PAL. 512
;3668   ;     locations are needed for CSR<22:14> to index into. The
;3669   ;     decoding logic supplies 10 control bits:
;3670   ;
;3671   ;                 ALU Source Select   (3)
;3672   ;                 ALU Function   (3)
;3673   ;                 ALU Result Destination   (3)
;3674   ;                 ALU Carry-In   (1)
;3675   ;
;3676   ;     The ROM addresses are therefore allocated in the following way:
;3677   ;
;3678   ;     Location     0 -   F       Incr. D-Bus, Output ALU, No Write.
;3679   ;                 10 -  1F       Incr. D-Bus, Output ALU, Write RAM.
;3680   ;                 20 -  3F       No-Op.
;3681   ;                 40 -  5F       Output WR, bypassing ALU.
;3682   ;                 60 -  7F       Pass D-Bus through ALU and write RAM.
;3683   ;                 80 -  BF       CSR<19:14> indicates 2901 function.
;3684   ;                 C0 -  FF       CSR<19:17> indicates 2901 function.
;3685   ;                100 - 1FF       CSR<21:16> indicates 2901 function.
;3686   ;
;3687   ;     For the detailed function table see the Microcode Listing.
```

```
;3688    .page
;3689    ;
;3690    ;
;3691    ;                          Table 2. Local Store Write
;3692    ;
;3693    ;     The Local Store Write Controller examines CSR<22:17>, CSR<6>, the
;3694    ;     SIZE  register  (SIZE<1:0>)  and the Compatibility Mode bit (CM).
;3695    ;     These eight bits are decoded to determine which parts of  the  LS
;3696    ;     to  write, if any.  The LS is written with data from the 2901 ALU
;3697    ;     (Y-Bus).
;3698    ;
;3699    ;                CSR      22  21  20  19  18  17
;3700    ;                         --  --  --  --  --  --
;3701    ;        1. BASIC          1   0   X   X   X   X    No Write
;3702    ;                          1   1   X   X   X   X    Write *
;3703    ;        2. MOVE           0   1   1   0   X   1    No Write
;3704    ;                          0   1   1   0   1   X    No Write
;3705    ;                          0   1   1   1   X   X    Write *
;3706    ;        3. EXTENDED       0   1   1   X   X   X    No Write
;3707    ;        4. MEM.REQ        0   0   1   1   X   X    No Write
;3708    ;        5. MISC           0   0   1   0   X   X    No Write
;3709    ;        6. JUMP           0   0   0   1   X   X    No Write
;3710    ;        7. DECODE         0   0   0   0   X   X    Conditional Write **
;3711    ;
;3712    ;  * LS Write, Data Type Dependent. These cases now examine CSR<6>.
;3713    ;
;3714    ;     If CSR<6> = 0, and CM = 0   Write LS<31:0>  (Data Type = LONG)
;3715    ;     If CSR<6> = 0, and CM = 1   Write LS<15:0>  (Compatibility
;3716    ;                                                  Mode. DT = WORD)
;3717    ;
;3718    ;     If CSR<6> = 1, the Write is conditional on the SIZE register.
;3719    ;
;3720    ;     If CM = 0   SIZE<1:0> = 00  Write LS<7:0>  (BYTE)
;3721    ;                           = 01  Write LS<15:0> (WORD)
;3722    ;                           = 1X  Write LS<31:0> (LONG)
;3723    ;
;3724    ;     If CM = 1  Write LS<15:0>  (WORD)
;3725    ;
;3726    ;  ** If CSR<8> = 0, No Write
;3727    ;        If CSR<8> = 1, Write LS<31:0>  (Writes incremented PC)
```

```
;3728   .page
;3729   ;
;3730   ;
;3731   ;                   Table 3. SKIP/JUMP Conditions
;3732   ;
;3733   ;This table summarizes the Skip and Jump conditions, and the other
;3734   ;special functions of the micro sequencer.
;3735   ;
;3736   ;SCTL codes 0-F, 17-1F are Skip conditions. The codes 10-17 execute
;3737   ;special functions. JCTL codes 0-B are Jump conditions, and C-F
;3738   ;execute special functions.
;3739   ;
;3740   ;
;3741   ;Sctl    Function                Sctl    Function
;3742   ;----    --------                ----    --------
;3743   ; 00     Not ALU N                10     RTN+1 if No Mem Err
;3744   ; 01     Not ALU Z                11     Loop if Not ALU Z
;3745   ; 02     Not ALU V                12     POP Stack
;3746   ; 03     ALU C                    13     Loop if ALU C
;3747   ; 04     Not PSL C                14     Return
;3748   ; 05     Branch False             15     No Skip (NOP)
;3749   ; 06     RDEST                    16     RTN+1
;3750   ; 07     Not Inter. Req.          17     Loop if No Inter. and No Sync
;3751   ; 08     ALU N                    18     Console ATTN
;3752   ; 09     ALU Z                    19     Console ACK
;3753   ; 0A     ALU V                    1A     Port Interrupt Pending
;3754   ; 0B     Not ALU C                1B     Reg. Backup Mask Flag
;3755   ; 0C     State 0                  1C     ALU N.XOR.ALU V
;3756   ; 0D     State 1                  1D     Not Memory Error
;3757   ; 0E     Not State 0              1E     Skip
;3758   ; 0F     Not State 1              1F     Sync
;3759   ;
;3760   ;
;3761   ;Jctl    Function                Jctl    Function
;3762   ;----    --------                ----    --------
;3763   ; 00     Not ALU N                08     ALU N
;3764   ; 01     Not ALU Z                09     ALU Z
;3765   ; 02     Not ALU V                0A     ALU V
;3766   ; 03     ALU C                    0B     Not ALU C
;3767   ; 04     JUMP                     0C     JSR
;3768   ; 05     Branch False             0D     JSR if IB Valid
;3769   ; 06     RDEST                    0E     No Jump; Skip if IB Valid
;3770   ; 07     Not Interr. Req          0F     IB Valid
;3771   ;
;3772
```

```
;3773   .PAGE  "2901 ALU Control Description"
;3774
;3775   ;
;3776   ;           The 2901 ALU is the heart of the CPU module.  It is
;3777   ;       responsible for all CPU calculations.  The control lines going
;3778   ;       to I0 through I8 control the function, input, and output of
;3779   ;       the 2901.  When a failure involving the 2901 occurs, the
;3780   ;       logical place to start looking is the control lines.  The
;3781   ;       following tables describe what the control signals should be
;3782   ;       for various functions.  Use it whenever the control lines are
;3783   ;       to be checked.  The test descriptions tell the operator what
;3784   ;       function the 2901 is supposed to be doing.
;3785   ;
;3786   ;           The SRC CTL on lines I0 through I2 (pins 12 through 14
;3787   ;       respectively) control where the inputs to the internal ALU
;3788   ;       within the 2901 come from.  The inputs to the internal ALU are
;3789   ;       labeled R and S.  R is one operand (4 bits) and S is the other
;3790   ;       operand (also 4 bits).  The R input can come from a Working
;3791   ;       Register (labeled A), the D bus (direct input), or be forced
;3792   ;       to 0.  The S input can come from either set of working
;3793   ;       registers (labeled A and B), the Q register, or forced to 0.
;3794   ;       A table of these control signals follow:
;3795   ;
;3796   ;
;3797   ;        ---------------------------
;3798   ;       |               |     | SRC   |
;3799   ;       |               |     |operands|
;3800   ;       |               |octal|-------|
;3801   ;       | I2  I1  I0    |code |  R  S |
;3802   ;       |---------------|-----|-------|
;3803   ;       | L   L   L     |  0  |  A  Q |
;3804   ;       | L   L   H     |  1  |  A  B |
;3805   ;       | L   H   L     |  2  |  0  Q |
;3806   ;       | L   H   H     |  3  |  0  B |
;3807   ;       | H   L   L     |  4  |  0  A |
;3808   ;       | H   L   H     |  5  |  D  A |
;3809   ;       | H   H   L     |  6  |  D  Q |
;3810   ;       | H   H   H     |  7  |  D  0 |
;3811   ;        ---------------------------
;3812   ;
;3813   ;         ALU SOURCE OPERAND CONTROL
;3814   ;
;3815   ;
;3816   ;
;3817   ;
;3818   ;           The ALU CTL on lines I3 through I5 (pins 26, 28, 27
;3819   ;       respectively) control what function the ALU is to perform on
;3820   ;       the two operands R and S.  A table follows:
```

```
;3821   .page
;3822   ;
;3823   ;
;3824   ;
;3825   ;
;3826   ;
;3827   ;
;3828   ;
;3829   ;
;3830   ;
;3831   ;
;3832   ;
;3833   ;
;3834   ;
;3835   ;
;3836   ;
;3837   ;
;3838   ;
;3839   ;
;3840   ;
;3841   ;
;3842   ;
;3843   ;
;3844   ;
;3845   ;
;3846   ;
;3847   ;
;3848   ;
;3849   ;
;3850   ;
;3851   ;
;3852   ;
;3853   ;
;3854   ;
;3855   ;
;3856   ;
;3857   ;
;3858   ;
;3859   ;
;3860   ;
;3861   ;
;3862   ;
;3863   ;
;3864   ;
;3865   ;
;3866   ;
;3867   ;
;3868   ;
;3869   ;
;3870   ;
;3871   ;
;3872   ;
;3873   ;
;3874   ;
```

I5	I4	I3	octal code	ALU function
L	L	L	0	R PLUS S
L	L	H	1	S MINUS R
L	H	L	2	R MINUS S
L	H	H	3	R OR S
H	L	L	4	R AND S
H	L	H	5	R NOT AND S
H	H	L	6	R XOR S
H	H	H	7	R XNOR S

ALU FUNCTION CONTROL

The DST CTL on lines I6 through I8 (pins 5, 7, 6 respectively) control what goes out onto the Y bus (either the internal ALU result or a WR directly) and what happens internal to the 2901 (shifting, writing the Q register etc). Internal shifting left or right is accomplished with this control. Below is a table explaining the various destination possibilities. The arrow refers to where the output will end up. B indicates the WR addressed by the B ADDR, F refers to the output of the internal ALU, and A refers to the output of the WR addressed by the A ADDR. Up is toward the MSB, while down is toward the LSB.

nomenclature in listing	I8	I7	I6	octal code	RAM function shift	RAM function load	Q-reg function shift	Q-reg function load	Y output
LOAD Q	L	L	L	0	NONE	NONE	NONE	F->Q	F
NOP	L	L	H	1	NONE	NONE	NONE	NONE	F
WRITE.B.A	L	H	L	2	NONE	F->B	NONE	NONE	A
WRITE.B.F	L	H	H	3	NONE	F->B	NONE	NONE	F
RSHF.RAM.Q	H	L	L	4	DOWN	F/2->B	DOWN	Q/2->Q	F
RSHF.RAM	H	L	H	5	DOWN	F/2->B	NONE	NONE	F
LSHF.RAM.Q	H	H	L	6	UP	2F->B	UP	2Q->Q	F
LSHF.RAM	H	H	H	7	UP	2F->B	NONE	NONE	F

ALU DESTINATION CONTROL

```
;3875   .page
;3876   .TOC    "DIAGNOSTIC MACROS"
;3877
;3878   LS.DATA.WORD/=<15:0>
;3879   UPPER.BYTE/=<23:16>
;3880
;3881   WORD[]          'UPPER.BYTE/0,LS.DATA.WORD/a1'
;3882   COUNT.LS[]      'UPPER.BYTE/0,LS.DATA.WORD/a1'
;3883   COUNT.MM[]      'UPPER.BYTE/1,LS.DATA.WORD/a1'
;3884   START.ADR[]     'UPPER.BYTE/0,LS.DATA.WORD/a1'
;3885
;3886
;3887   ASCII.LIT/=<15:0>
;3888
;3889           CA=4341         ;ASCII VALUE FOR FILE NAMES
;3890           CB=4342         ;   |
;3891           CC=4343         ;   |
;3892           CD=4344         ;   |
;3893           CE=4345         ;   V
;3894           CF=4346
;3895           CG=4347
;3896           CH=4348
;3897
;3898   ASCII []        'UPPER.BYTE/0,ASCII.LIT/a1'
;3899
;3900   .REGION/0,6F
```

```
                    ;3901   .PAGE   "TEST POINTERS"
                    ;3902   ;
                    ;3903   ;       This portion contains the pointers to all tests in this section.  The
                    ;3904   ;       WCS address of the JUMP instruction corresponds to the test number.
                    ;3905   ;       E.g. WCS 5 contains a JUMP to test 5.  All unused test numbers will
                    ;3906   ;       contain a JUMP to an error routine.  This portion is 80. locations
                    ;3907   ;       long, allowing for up to 80. tests per section, from WCS address 1 to
                    ;3908   ;       WCS address 4F.
                    ;3909   ;
                    ;3910   1:                                      ;TEST POINTERS BEGIN AT WCS ADDRESS 1
                    ;3911
U 0001, 0870,04     ;3912           JMP [T.1]                       ;GO TO TEST 1
U 0002, 8873,24     ;3913           JMP [T.2]                       ;GO TO TEST 2
U 0003, 0878,14     ;3914           JMP [T.3]                       ;GO TO TEST 3
U 0004, 087A,F4     ;3915           JMP [T.4]                       ;GO TO TEST 4
U 0005, 887D,C4     ;3916           JMP [T.5]                       ;GO TO TEST 5
U 0006, 8880,14     ;3917           JMP [T.6]                       ;GO TO TEST 6
U 0007, 0882,44     ;3918           JMP [T.7]                       ;GO TO TEST 7
U 0008, 0883,94     ;3919           JMP [T.8]                       ;GO TO TEST 8
U 0009, 0885,34     ;3920           JMP [T.9]                       ;GO TO TEST 9
U 000A, 0886,64     ;3921           JMP [T.A]                       ;GO TO TEST A
U 000B, 0888,74     ;3922           JMP [T.B]                       ;GO TO TEST B
U 000C, 888C,E4     ;3923           JMP [T.C]                       ;GO TO TEST C
U 000D, 8891,84     ;3924           JMP [T.D]                       ;GO TO TEST D
U 000E, 889B,D4     ;3925           JMP [T.E]                       ;GO TO TEST E
U 000F, 88A8,D4     ;3926           JMP [T.F]                       ;GO TO TEST F
U 0010, 08A9,E4     ;3927           JMP [T.10]                      ;GO TO TEST 10
U 0011, 88AA,F4     ;3928           JMP [T.11]                      ;GO TO TEST 11
U 0012, 08BA,64     ;3929           JMP [T.12]                      ;GO TO TEST 12
U 0013, 08C7,C4     ;3930           JMP [T.13]                      ;GO TO TEST 13
U 0014, 88CA,C4     ;3931           JMP [T.14]                      ;GO TO TEST 14
U 0015, 88D1,34     ;3932           JMP [T.15]                      ;GO TO TEST 15
U 0016, 88E4,54     ;3933           JMP [T.16]                      ;GO TO TEST 16
U 0017, 88EC,14     ;3934           JMP [T.17]                      ;GO TO TEST 17
U 0018, 88ED,C4     ;3935           JMP [T.18]                      ;GO TO TEST 18
U 0019, 88EF,84     ;3936           JMP [T.19]                      ;GO TO TEST 19
U 001A, 88F7,24     ;3937           JMP [T.1A]                      ;GO TO TEST 1A
U 001B, 88FB,24     ;3938           JMP [T.1B]                      ;GO TO TEST 1B
U 001C, 0900,04     ;3939           JMP [T.1C]                      ;GO TO TEST 1C
U 001D, 0906,64     ;3940           JMP [T.1D]                      ;GO TO TEST 1D
U 001E, 890B,A4     ;3941           JMP [T.1E]                      ;GO TO TEST 1E
U 001F, 0911,F4     ;3942           JMP [T.1F]                      ;GO TO TEST 1F
U 0020, 8913,F4     ;3943           JMP [T.20]                      ;GO TO TEST 20
U 0021, 0918,64     ;3944           JMP [T.21]                      ;GO TO TEST 21
U 0022, 0927,94     ;3945           JMP [T.22]                      ;GO TO TEST 22
U 0023, 0929,24     ;3946           JMP [T.23]                      ;GO TO TEST 23
U 0024, 892C,F4     ;3947           JMP [T.24]                      ;GO TO TEST 24
U 0025, 093C,64     ;3948           JMP [T.25]                      ;GO TO TEST 25
U 0026, 8944,84     ;3949           JMP [T.26]                      ;GO TO TEST 26
U 0027, 8947,84     ;3950           JMP [T.27]                      ;GO TO TEST 27
U 0028, 094D,A4     ;3951           JMP [T.28]                      ;GO TO TEST 28
U 0029, 0951,44     ;3952           JMP [T.29]                      ;GO TO TEST 29
U 002A, 0956,C4     ;3953           JMP [T.2A]                      ;GO TO TEST 2A
U 002B, 8958,A4     ;3954           JMP [T.2B]                      ;GO TO TEST 2B
U 002C, 8960,84     ;3955           JMP [T.2C]                      ;GO TO TEST 2C
```

```
U 002D, 096C,A4 ;3956          JMP [T.2D]                ;GO TO TEST 2D
U 002E, 0971,94 ;3957          JMP [T.2E]                ;GO TO TEST 2E
U 002F, 0976,74 ;3958          JMP [T.2F]                ;GO TO TEST 2F
U 0030, 8979,64 ;3959          JMP [T.30]                ;GO TO TEST 30
U 0031, 8985,64 ;3960          JMP [T.31]                ;GO TO TEST 31
U 0032, 0995,94 ;3961          JMP [T.32]                ;GO TO TEST 32
U 0033, 09AB,E4 ;3962          JMP [T.33]                ;GO TO TEST 33
U 0034, 89BC,C4 ;3963          JMP [T.34]                ;GO TO TEST 34
U 0035, 89BF,04 ;3964          JMP [T.35]                ;GO TO TEST 35
U 0036, 89C5,B4 ;3965          JMP [T.36]                ;GO TO TEST 36
U 0037, 89C7,94 ;3966          JMP [T.37]                ;GO TO TEST 37
U 0038, 09CE,74 ;3967          JMP [T.38]                ;GO TO TEST 38
U 0039, 09D4,A4 ;3968          JMP [T.39]                ;GO TO TEST 39
U 003A, 89DB,B4 ;3969          JMP [T.3A]                ;GO TO TEST 3A
U 003B, 89E0,C4 ;3970          JMP [T.3B]                ;GO TO TEST 3B
U 003C, 89E5,C4 ;3971          JMP [T.3C]                ;GO TO TEST 3C
U 003D, 89F1,A4 ;3972          JMP [T.3D]                ;GO TO TEST 3D
U 003E, 89F5,84 ;3973          JMP [T.3E]                ;GO TO TEST 3E
U 003F, 09FE,14 ;3974          JMP [T.3F]                ;GO TO TEST 3F
U 0040, 0A05,A4 ;3975          JMP [T.40]                ;GO TO TEST 40
U 0041, 8A06,D4 ;3976          JMP [T.41]                ;GO TO TEST 41
U 0042, 0A09,34 ;3977          JMP [T.42]                ;GO TO TEST 42
U 0043, 8A11,74 ;3978          JMP [T.43]                ;GO TO TEST 44
U 0044, 8A15,C4 ;3979          JMP [READ.UBE.REGS]       ;GO TO ROUTINE TO READ UBE REGS
                ;3980
U 0045, 886B,E4 ;3981          JMP [ERR.NO.TEST]         ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                ;3982                                    ; IN THIS OVERLAY
U 0046, 886B,E4 ;3983          JMP [ERR.NO.TEST]         ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                ;3984                                    ; IN THIS OVERLAY
U 0047, 886B,E4 ;3985          JMP [ERR.NO.TEST]         ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                ;3986                                    ; IN THIS OVERLAY
U 0048, 886B,E4 ;3987          JMP [ERR.NO.TEST]         ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                ;3988                                    ; IN THIS OVERLAY
U 0049, 886B,E4 ;3989          JMP [ERR.NO.TEST]         ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                ;3990                                    ; IN THIS OVERLAY
U 004A, 886B,E4 ;3991          JMP [ERR.NO.TEST]         ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                ;3992                                    ; IN THIS OVERLAY
U 004B, 886B,E4 ;3993          JMP [ERR.NO.TEST]         ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                ;3994                                    ; IN THIS OVERLAY
U 004C, 886B,E4 ;3995          JMP [ERR.NO.TEST]         ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                ;3996                                    ; IN THIS OVERLAY
U 004D, 886B,E4 ;3997          JMP [ERR.NO.TEST]         ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                ;3998                                    ; IN THIS OVERLAY
U 004E, 886B,E4 ;3999          JMP [ERR.NO.TEST]         ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                ;4000                                    ; IN THIS OVERLAY
U 004F, 886B,E4 ;4001          JMP [ERR.NO.TEST]         ;GO TO ERROR ROUTINE. NO SUCH TEST NUMBER
                ;4002                                    ; IN THIS OVERLAY
```

```
                    ;4003   .PAGE   'DIAGNOSTIC POINTERS'
                    ;4004   ;
                    ;4005   ;       This section contains all the pointers needed for this diagnostic
                    ;4006   ;       overlay.  The first pointer (at WCS location 0) points to a data
                    ;4007   ;       transfer routine.  It is the first location executed after a new
                    ;4008   ;       WCS load.  The insruction here is a JUMP to TRANSFER.POINT which may
                    ;4009   ;       contain instructions to transfer data.  If no data is to be trans-
                    ;4010   ;       ferred, or at the completion of the data transfers, the CPU will
                    ;4011   ;       issue CPU ATTN with all bits of the control and status word clear.
                    ;4012   ;
                    ;4013   ;       The next 80. locations (from WCS location 1 to WCS location 4F)
                    ;4014   ;       contain pointers to each test in this overlay.  The pointers are
                    ;4015   ;       JUMP instructions to the test number corresponding to the location
                    ;4016   ;       number.   .g. location 5 will contain a JUMP to test 5.  All unused
                    ;4017   ;       test numbers will contain a JUMP to an error routine.  This portion
                    ;4018   ;       appears after the definitions in this listing.
                    ;4019   ;
                    ;4020   ;       Finally the next 32. locations (from WCS location 50 to 6F) contain
                    ;4021   ;       pointers (Jump instructions) to WCS subroutines which are to be used by
                    ;4022   ;       the console diagnostic monitor.
                    ;4023   ;
U 0000, 086C,24     ;4024   0:      JMP [TRANSFER.POINT]            ;GO TO ROUTINE THAT LOADS LS 7 WITH
                    ;4025                                           ; POINTER TO DATA SECTION AND ISSUES
                    ;4026                                           ; CPU ATTN WITH TRANSFER BIT SET.
                    ;4027
                    ;4028   50:                                     ;START AT WCS LOCATION 50 FOR SUB-
                    ;4029                                           ; ROUTINE POINTERS
                    ;4030   ;
                    ;4031   ; SUBROUTINE POINTERS
                    ;4032   ;
U 0050, 0860,84     ;4033           JMP [DEPOSIT.CSR]               ;GO TO WCS SUBROUTINE WHICH WRITES THE
                    ;4034                                           ; MCT CSR FOR DEPOSIT CSR COMMAND
U 0051, 0860,D4     ;4035           JMP [EXAMINE.CSR]               ;GO TO WCS SUBROUTINE WHICH READS THE
                    ;4036                                           ; MCT CSR FOR EXAMINE CSR COMMAND
U 0052, 0861,F4     ;4037           JMP [DEPOSIT.TB]                ;GO TO WCS SUBROUTINE WHICH WRITES THE
                    ;4038                                           ; MCT TRANSLATION BUFFER FOR DEPOSIT
                    ;4039                                           ; TB COMMAND
U 0053, 8863,A4     ;4040           JMP [EXAMINE.TB]                ;GO TO WCS SUBROUTINE WHICH READS THE
                    ;4041                                           ; MCT TRANSLATION BUFFER FOR EXAMINE
                    ;4042                                           ; TB COMMAND
U 0054, 8865,C4     ;4043           JMP [DEPOSIT.MM]                ;GO TO WCS SUBROUTINE WHICH WRITES MAIN
                    ;4044                                           ; MEMORY FOR DEPOSIT MM COMMAND.  ALSO
                    ;4045                                           ; USED TRANSFERING DATA FROM WCS TO
                    ;4046                                           ; MAIN MEMORY.
U 0055, 0866,24     ;4047           JMP [EXAMINE.MM]                ;GO TO WCS SUBROUTINE WHICH READS MAIN
                    ;4048                                           ; MEMORY FOR EXAMINE MM COMMAND.
U 0056, 0868,94     ;4049           JMP [SAVE.WR]                   ;GO TO WCS SUBROUTINE WHICH STORES THE
                    ;4050                                           ; CONTENTS OF WR0-WR3 IN LS 0-3
U 0057, 8868,E4     ;4051           JMP [RESTORE.WR]                ;GO TO WCS SUBROUTINE WHICH RESTORES THE
                    ;4052                                           ; CONTENTS OF WR0-WR3 FROM LS 0-3
U 0058, 8864,24     ;4053           JMP [DEPOSIT.UBS]               ;GO TO WCS SUBROUTINE WHICH WRITES THE
                    ;4054                                           ; UNIBUS MAP ON THE MEMORY CONTROLLER
U 0059, 0865,44     ;4055           JMP [EXAMINE.UBS]               ;GO TO WCS SUBROUTINE WHICH READS THE
                    ;4056                                           ; UNIBUS MAP ON THE MEMORY CONTROLLER
U 005A, 0866,84     ;4057           JMP [EXAMINE.ICSR]              ;GO TO WCS SUBROUTINE WHICH READS THE
```

```
                ;4058                                   ; IDC CSR
U 005B, 0866,B4 ;4059           JMP [EXAMINE.IDAR]      ;GO TO WCS SUBROUTINE WHICH READS THE
                ;4060                                   ; IDC DAR
U 005C, 0866,E4 ;4061           JMP [EXAMINE.DBUF]      ;GO TO WCS SUBROUTINE WHICH READS THE
                ;4062                                   ; IDC DBUF
U 005D, 8867,14 ;4063           JMP [EXAMINE.PATT]      ;GO TO WCS SUBROUTINE WHICH READS THE
                ;4064                                   ; IDC ECC PATTERN
U 005E, 8867,44 ;4065           JMP [EXAMINE.POSIT]     ;GO TO WCS SUBROUTINE WHICH READS THE
                ;4066                                   ; IDC ECC POSITION
U 005F, 0867,94 ;4067           JMP [DEPOSIT.ICSR]      ;GO TO WCS SUBROUTINE WHICH WRITES THE
                ;4068                                   ; IDC CSR
U 0060, 0867,C4 ;4069           JMP [DEPOSIT.IDAR]      ;GO TO WCS SUBROUTINE WHICH WRITES THE
                ;4070                                   ; IDC DAR
U 0061, 0867,F4 ;4071           JMP [DEPOSIT.DBUF]      ;GO TO WCS SUBROUTINE WHICH WRITES THE
                ;4072                                   ; IDC DBUF
U 0062, 8868,24 ;4073           JMP [CLEAR.FIFO.ADDR]   ;GO TO WCS THE SUBROUTINE WHICH CLEARS
                ;4074                                   ; THE IDC FIFO ADDRESS
U 0063, 8868,44 ;4075           JMP [SELECT.FIFO.A]     ;GO TO THE WCS SUBROUTINE WHICH SELECTS
                ;4076                                   ; FIFO A
U 0064, 0868,64 ;4077           JMP [SELECT.FIFO.B]     ;GO TO THE WCS SUBROUTINE WHICH SELECTS
                ;4078                                   ; FIFO B
U 0065, DB00,15 ;4079           NOP                     ;NOT USED
U 0066, DB00,15 ;4080           NOP                     ;NOT USED
U 0067, DB00,15 ;4081           NOP                     ;NOT USED
U 0068, DB00,15 ;4082           NOP                     ;NOT USED
U 0069, DB00,15 ;4083           NOP                     ;NOT USED
U 006A, DB00,15 ;4084           NOP                     ;NOT USED
U 006B, DB00,15 ;4085           NOP                     ;NOT USED
U 006C, DB00,15 ;4086           NOP                     ;NOT USED
U 006D, DB00,15 ;4087           NOP                     ;NOT USED
U 006E, DB00,15 ;4088           NOP                     ;NOT USED
U 006F, DB00,15 ;4089           NOP                     ;NOT USED
```

```
                    ;4090   .PAGE
                    ;4091   .REGION/70,5FF
                    ;4092   .TOC    "LS DATA SECTION"
                    ;4093
                    ;4094   ;+++
                    ;4095   ;       This section lists the data that will be transfered to LS by the con-
                    ;4096   ;       sole as part of the initialization performed in test 0.  The TRANSFER
                    ;4097   ;       DATA routine will point to this data area.  LS contains the constants,
                    ;4098   ;       control and status word, and temporary storage locations used by the
                    ;4099   ;       tests and subroutines of the microdiagnostic.
                    ;4100   ;         The LS is 32 bits wide.  Each of these words is 16 bits, so two con-
                    ;4101   ;       secutive words listed here will be loaded into each consecutive LS loc-
                    ;4102   ;       ation.  The first word listed will be bits 0-15 of the LS location, the
                    ;4103   ;       second word listed will be bits 16-31 of the same LS location.
                    ;4104   ;         The locations 78-7F in the first half of LS and F8-FF in the second
                    ;4105   ;       half of LS are not actual LS locations.  They force an access to one of
                    ;4106   ;       several registers in the CPU or force an indexing feature to another LS
                    ;4107   ;       location.  For that reason, they are not written via the transfer rou-
                    ;4108   ;       tine.
                    ;4109   ;         Note:  This table is in hexadecimal format i.e. each digit represents
                    ;4110   ;       four bits, so four digits represnts a full 16 bit word.  The micro-
                    ;4111   ;       assembler requires that a hexidecimal value that starts with a letter
                    ;4112   ;       (e.g. FFFF) be preceeded by a 0 (0FFFF).  So some table values will
                    ;4113   ;       contain 5 hexidecimal digits.  The fifth digit is not actually used.
                    ;4114   ;
                    ;4115   ;---
                    ;4116
                    ;4117   TRANSFER.DATA.LS1:
                    ;4118
U 0070, 0000,78     ;4119           COUNT.LS[0078]          ;LONGWORD COUNT (NUMBER OF LONGWORDS TO TRANS-
                    ;4120                                   ; FER TO LS = 120 DECIMAL) DESTINATION IS LS
U 0071, 0000,00     ;4121           START.ADR[0000]         ;DESTINATION START ADDRESS (START AT LS 0)
                    ;4122
U 0072, 0000,00     ;4123           WORD[0000]              ;LOCATION 0
U 0073, 0000,00     ;4124           WORD[0000]
                    ;4125
U 0074, 0000,00     ;4126           WORD[0000]              ;LOCATION 1
U 0075, 0000,00     ;4127           WORD[0000]
                    ;4128
U 0076, 0000,00     ;4129           WORD[0000]              ;LOCATION 2
U 0077, 0000,00     ;4130           WORD[0000]
                    ;4131
U 0078, 0000,00     ;4132           WORD[0000]              ;LOCATION 3
U 0079, 0000,00     ;4133           WORD[0000]
                    ;4134
U 007A, 0000,00     ;4135           WORD[0000]              ;LOCATION 4
U 007B, 0000,00     ;4136           WORD[0000]
                    ;4137
U 007C, 0000,00     ;4138           WORD[0000]              ;LOCATION 5
U 007D, 0000,00     ;4139           WORD[0000]
                    ;4140
U 007E, 0000,00     ;4141           WORD[0000]              ;LOCATION 6
U 007F, 0000,00     ;4142           WORD[0000]
                    ;4143
U 0080, 0000,00     ;4144           WORD[0000]              ;LOCATION 7
```

```
U 0081, 0000,00 ;4145          WORD[0000]
                ;4146
U 0082, 0000,00 ;4147          WORD[0000]              ;LOCATION 8
U 0083, 0000,00 ;4148          WORD[0000]
                ;4149
U 0084, 0000,00 ;4150          WORD[0000]              ;LOCATION 9
U 0085, 0000,00 ;4151          WORD[0000]
                ;4152
U 0086, 0000,00 ;4153          WORD[0000]              ;LOCATION 0A
U 0087, 0000,00 ;4154          WORD[0000]
                ;4155
U 0088, 0000,00 ;4156          WORD[0000]              ;LOCATION 0B
U 0089, 0000,00 ;4157          WORD[0000]
                ;4158
U 008A, 0000,00 ;4159          WORD[0000]              ;LOCATION 0C
U 008B, 0000,00 ;4160          WORD[0000]
                ;4161
U 008C, 0000,00 ;4162          WORD[0000]              ;LOCATION 0D
U 008D, 0000,00 ;4163          WORD[0000]
                ;4164
U 008E, 0000,00 ;4165          WORD[0000]              ;LOCATION 0E
U 008F, 0000,00 ;4166          WORD[0000]
                ;4167
U 0090, 0000,00 ;4168          WORD[0000]              ;LOCATION 0F
U 0091, 0000,00 ;4169          WORD[0000]
                ;4170
U 0092, 0000,00 ;4171          WORD[0000]              ;LOCATION 10
U 0093, 0000,00 ;4172          WORD[0000]
                ;4173
U 0094, 0000,00 ;4174          WORD[0000]              ;LOCATION 11
U 0095, 0000,00 ;4175          WORD[0000]
                ;4176
U 0096, 0000,00 ;4177          WORD[0000]              ;LOCATION 12
U 0097, 0000,00 ;4178          WORD[0000]
                ;4179
U 0098, 0000,FF ;4180          WORD[00FF]              ;LOCATION 13
U 0099, 0000,00 ;4181          WORD[0000]
                ;4182
U 009A, 00FF,FF ;4183          WORD[0FFFF]             ;LOCATION 14
U 009B, 0000,00 ;4184          WORD[0000]
                ;4185
U 009C, 0000,00 ;4186          WORD[0000]              ;LOCATION 15
U 009D, 00FF,00 ;4187          WORD[0FF00]
                ;4188
U 009E, 00FF,00 ;4189          WORD[0FF00]             ;LOCATION 16
U 009F, 00FF,FF ;4190          WORD[0FFFF]
                ;4191
U 00A0, 003F,FF ;4192          WORD[3FFF]              ;LOCATION 17
U 00A1, 0001,00 ;4193          WORD[0100]
                ;4194
U 00A2, 003F,FF ;4195          WORD[3FFF]              ;LOCATION 18
U 00A3, 007F,FE ;4196          WORD[7FFE]
                ;4197
U 00A4, 00FF,FF ;4198          WORD[0FFFF]             ;LOCATION 19
U 00A5, 00FE,7F ;4199          WORD[0FE7F]
```

```
                    ;4200
U 00A6, 0000,00     ;4201           WORD[0000]              ;LOCATION 1A
U 00A7, 007F,F8     ;4202           WORD[7FF8]
                    ;4203
U 00A8, 0080,00     ;4204           WORD[8000]              ;LOCATION 1B
U 00A9, 007F,FF     ;4205           WORD[7FFF]
                    ;4206
U 00AA, 0000,00     ;4207           WORD[0000]              ;LOCATION 1C
U 00AB, 0000,00     ;4208           WORD[0000]
                    ;4209
U 00AC, 0000,00     ;4210           WORD[0000]              ;LOCATION 1D
U 00AD, 0000,00     ;4211           WORD[0000]
                    ;4212
U 00AE, 0005,00     ;4213           WORD[0500]              ;LOCATION 1E
U 00AF, 0000,80     ;4214           WORD[0080]
                    ;4215
U 00B0, 0005,00     ;4216           WORD[0500]              ;LOCATION 1F
U 00B1, 0000,00     ;4217           WORD[0000]
                    ;4218
U 00B2, 0000,01     ;4219           WORD[0001]              ;LOCATION 20
U 00B3, 0000,00     ;4220           WORD[0000]
                    ;4221
U 00B4, 0000,02     ;4222           WORD[0002]              ;LOCATION 21
U 00B5, 0000,00     ;4223           WORD[0000]
                    ;4224
U 00B6, 0000,04     ;4225           WORD[0004]              ;LOCATION 22
U 00B7, 0000,00     ;4226           WORD[0000]
                    ;4227
U 00B8, 0000,08     ;4228           WORD[0008]              ;LOCATION 23
U 00B9, 0000,00     ;4229           WORD[0000]
                    ;4230
U 00BA, 0000,10     ;4231           WORD[0010]              ;LOCATION 24
U 00BB, 0000,00     ;4232           WORD[0000]
                    ;4233
U 00BC, 0000,20     ;4234           WORD[0020]              ;LOCATION 25
U 00BD, 0000,00     ;4235           WORD[0000]
                    ;4236
U 00BE, 0000,40     ;4237           WORD[0040]              ;LOCATION 26
U 00BF, 0000,00     ;4238           WORD[0000]
                    ;4239
U 00C0, 0000,80     ;4240           WORD[0080]              ;LOCATION 27
U 00C1, 0000,00     ;4241           WORD[0000]
                    ;4242
U 00C2, 0001,00     ;4243           WORD[0100]              ;LOCATION 28
U 00C3, 0000,00     ;4244           WORD[0000]
                    ;4245
U 00C4, 0002,00     ;4246           WORD[0200]              ;LOCATION 29
U 00C5, 0000,00     ;4247           WORD[0000]
                    ;4248
U 00C6, 0004,00     ;4249           WORD[0400]              ;LOCATION 2A
U 00C7, 0000,00     ;4250           WORD[0000]
                    ;4251
U 00C8, 0008,00     ;4252           WORD[0800]              ;LOCATION 2B
U 00C9, 0000,00     ;4253           WORD[0000]
                    ;4254
```

```
U 00CA, 0010,00 ;4255          WORD[1000]          ;LOCATION 2C
U 00CB, 0000,00 ;4256          WORD[0000]
                ;4257
U 00CC, 0020,00 ;4258          WORD[2000]          ;LOCATION 2D
U 00CD, 0000,00 ;4259          WORD[0000]
                ;4260
U 00CE, 0040,00 ;4261          WORD[4000]          ;LOCATION 2E
U 00CF, 0000,00 ;4262          WORD[0000]
                ;4263
U 00D0, 0080,00 ;4264          WORD[8000]          ;LOCATION 2F
U 00D1, 0000,00 ;4265          WORD[0000]
                ;4266
U 00D2, 0000,00 ;4267          WORD[0000]          ;LOCATION 30
U 00D3, 0000,01 ;4268          WORD[0001]
                ;4269
U 00D4, 0000,00 ;4270          WORD[0000]          ;LOCATION 31
U 00D5, 0000,02 ;4271          WORD[0002]
                ;4272
U 00D6, 0000,00 ;4273          WORD[0000]          ;LOCATION 32
U 00D7, 0000,04 ;4274          WORD[0004]
                ;4275
U 00D8, 0000,00 ;4276          WORD[0000]          ;LOCATION 33
U 00D9, 0000,08 ;4277          WORD[0008]
                ;4278
U 00DA, 0000,00 ;4279          WORD[0000]          ;LOCATION 34
U 00DB, 0000,10 ;4280          WORD[0010]
                ;4281
U 00DC, 0000,00 ;4282          WORD[0000]          ;LOCATION 35
U 00DD, 0000,20 ;4283          WORD[0020]
                ;4284
U 00DE, 0000,00 ;4285          WORD[0000]          ;LOCATION 36
U 00DF, 0000,40 ;4286          WORD[0040]
                ;4287
U 00E0, 0000,00 ;4288          WORD[0000]          ;LOCATION 37
U 00E1, 0000,80 ;4289          WORD[0080]
                ;4290
U 00E2, 0000,00 ;4291          WORD[0000]          ;LOCATION 38
U 00E3, 0001,00 ;4292          WORD[0100]
                ;4293
U 00E4, 0000,00 ;4294          WORD[0000]          ;LOCATION 39
U 00E5, 0002,00 ;4295          WORD[0200]
                ;4296
U 00E6, 0000,00 ;4297          WORD[0000]          ;LOCATION 3A
U 00E7, 0004,00 ;4298          WORD[0400]
                ;4299
U 00E8, 0000,00 ;4300          WORD[0000]          ;LOCATION 3B
U 00E9, 0008,00 ;4301          WORD[0800]
                ;4302
U 00EA, 0000,00 ;4303          WORD[0000]          ;LOCATION 3C
U 00EB, 0010,00 ;4304          WORD[1000]
                ;4305
U 00EC, 0000,00 ;4306          WORD[0000]          ;LOCATION 3D
U 00ED, 0020,00 ;4307          WORD[2000]
                ;4308
U 00EE, 0000,00 ;4309          WORD[0000]          ;LOCATION 3E
```

```
U 00EF, 0040,00 ;4310           WORD[4000]
                ;4311
U 00F0, 0000,00 ;4312           WORD[0000]              ;LOCATION 3F
U 00F1, 0080,00 ;4313           WORD[8000]
                ;4314
U 00F2, 0000,00 ;4315           WORD[0000]              ;LOCATION 40
U 00F3, 0000,00 ;4316           WORD[0000]
                ;4317
U 00F4, 0000,00 ;4318           WORD[0000]              ;LOCATION 41
U 00F5, 0000,00 ;4319           WORD[0000]
                ;4320
U 00F6, 0000,00 ;4321           WORD[0000]              ;LOCATION 42
U 00F7, 0000,00 ;4322           WORD[0000]
                ;4323
U 00F8, 0000,00 ;4324           WORD[0000]              ;LOCATION 43
U 00F9, 0000,00 ;4325           WORD[0000]
                ;4326
U 00FA, 0000,00 ;4327           WORD[0000]              ;LOCATION 44
U 00FB, 0000,00 ;4328           WORD[0000]
                ;4329
U 00FC, 0000,00 ;4330           WORD[0000]              ;LOCATION 45
U 00FD, 0000,00 ;4331           WORD[0000]
                ;4332
U 00FE, 0000,00 ;4333           WORD[0000]              ;LOCATION 46
U 00FF, 0000,00 ;4334           WORD[0000]
                ;4335
U 0100, 0000,00 ;4336           WORD[0000]              ;LOCATION 47
U 0101, 0000,00 ;4337           WORD[0000]
                ;4338
U 0102, 0000,00 ;4339           WORD[0000]              ;LOCATION 48
U 0103, 0000,00 ;4340           WORD[0000]
                ;4341
U 0104, 0000,00 ;4342           WORD[0000]              ;LOCATION 49
U 0105, 0000,00 ;4343           WORD[0000]
                ;4344
U 0106, 0000,00 ;4345           WORD[0000]              ;LOCATION 4A
U 0107, 0000,00 ;4346           WORD[0000]
                ;4347
U 0108, 0000,00 ;4348           WORD[0000]              ;LOCATION 4B
U 0109, 0000,00 ;4349           WORD[0000]
                ;4350
U 010A, 00AA,AA ;4351           WORD[0AAAA]             ;LOCATION 4C
U 010B, 00AA,AA ;4352           WORD[0AAAA]
                ;4353
U 010C, 0055,55 ;4354           WORD[5555]              ;LOCATION 4D
U 010D, 0055,55 ;4355           WORD[5555]
                ;4356
U 010E, 0000,00 ;4357           WORD[0000]              ;LOCATION 4E
U 010F, 0000,00 ;4358           WORD[0000]
                ;4359
U 0110, 00FF,FF ;4360           WORD[0FFFF]             ;LOCATION 4F
U 0111, 00FF,FF ;4361           WORD[0FFFF]
                ;4362
U 0112, 0000,00 ;4363           WORD[0000]              ;LOCATION 50
U 0113, 0000,00 ;4364           WORD[0000]
```

```
                    ;4365
U 0114, 0000,00     ;4366           WORD[0000]              ;LOCATION 51
U 0115, 0000,00     ;4367           WORD[0000]
                    ;4368
U 0116, 0000,00     ;4369           WORD[0000]              ;LOCATION 52
U 0117, 0000,00     ;4370           WORD[0000]
                    ;4371
U 0118, 0000,00     ;4372           WORD[0000]              ;LOCATION 53
U 0119, 0000,00     ;4373           WORD[0000]
                    ;4374
U 011A, 0000,00     ;4375           WORD[0000]              ;LOCATION 54
U 011B, 0000,00     ;4376           WORD[0000]
                    ;4377
U 011C, 0000,00     ;4378           WORD[0000]              ;LOCATION 55
U 011D, 0000,00     ;4379           WORD[0000]
                    ;4380
U 011E, 0000,00     ;4381           WORD[0000]              ;LOCATION 56
U 011F, 0000,00     ;4382           WORD[0000]
                    ;4383
U 0120, 0000,00     ;4384           WORD[0000]              ;LOCATION 57
U 0121, 0000,00     ;4385           WORD[0000]
                    ;4386
U 0122, 0000,00     ;4387           WORD[0000]              ;LOCATION 58
U 0123, 0000,00     ;4388           WORD[0000]
                    ;4389
U 0124, 0000,00     ;4390           WORD[0000]              ;LOCATION 59
U 0125, 0000,00     ;4391           WORD[0000]
                    ;4392
U 0126, 00F0,00     ;4393           WORD[0F000]             ;LOCATION 5A
U 0127, 00FF,FF     ;4394           WORD[0FFFF]
                    ;4395
U 0128, 00F0,02     ;4396           WORD[0F002]             ;LOCATION 5B
U 0129, 00FF,FF     ;4397           WORD[0FFFF]
                    ;4398
U 012A, 00F0,04     ;4399           WORD[0F004]             ;LOCATION 5C
U 012B, 00FF,FF     ;4400           WORD[0FFFF]
                    ;4401
U 012C, 00F0,06     ;4402           WORD[0F006]             ;LOCATION 5D
U 012D, 00FF,FF     ;4403           WORD[0FFFF]
                    ;4404
U 012E, 00F0,08     ;4405           WORD[0F008]             ;LOCATION 5E
U 012F, 00FF,FF     ;4406           WORD[0FFFF]
                    ;4407
U 0130, 00F0,0E     ;4408           WORD[0F00E]             ;LOCATION 5F
U 0131, 00FF,FF     ;4409           WORD[0FFFF]
                    ;4410
U 0132, 0000,03     ;4411           WORD[0003]              ;LOCATION 60
U 0133, 0000,00     ;4412           WORD[0000]
                    ;4413
U 0134, 0000,12     ;4414           WORD[0012]              ;LOCATION 61
U 0135, 0000,00     ;4415           WORD[0000]
                    ;4416
U 0136, 0033,33     ;4417           WORD[3333]              ;LOCATION 62
U 0137, 0033,33     ;4418           WORD[3333]
                    ;4419
```

```
U 0138, 000F,0F ;4420          WORD[0F0F]          ;LOCATION 63
U 0139, 000F,0F ;4421          WORD[0F0F]
                ;4422
U 013A, 0000,FF ;4423          WORD[00FF]          ;LOCATION 64
U 013B, 0000,FF ;4424          WORD[00FF]
                ;4425
U 013C, 0001,62 ;4426          WORD[0162]          ;LOCATION 65
U 013D, 0000,00 ;4427          WORD[0000]
                ;4428
U 013E, 0000,28 ;4429          WORD[0028]          ;LOCATION 66
U 013F, 0000,00 ;4430          WORD[0000]
                ;4431
U 0140, 0000,0C ;4432          WORD[000C]          ;LOCATION 67
U 0141, 0000,00 ;4433          WORD[0000]
                ;4434
U 0142, 0000,00 ;4435          WORD[0000]          ;LOCATION 68
U 0143, 0000,00 ;4436          WORD[0000]
                ;4437
U 0144, 0000,00 ;4438          WORD[0000]          ;LOCATION 69
U 0145, 0000,00 ;4439          WORD[0000]
                ;4440
U 0146, 0000,00 ;4441          WORD[0000]          ;LOCATION 6A
U 0147, 0000,00 ;4442          WORD[0000]
                ;4443
U 0148, 0000,00 ;4444          WORD[0000]          ;LOCATION 6B
U 0149, 0000,00 ;4445          WORD[0000]
                ;4446
U 014A, 0000,00 ;4447          WORD[0000]          ;LOCATION 6C
U 014B, 0000,00 ;4448          WORD[0000]
                ;4449
U 014C, 0000,00 ;4450          WORD[0000]          ;LOCATION 6D
U 014D, 0000,00 ;4451          WORD[0000]
                ;4452
U 014E, 0000,00 ;4453          WORD[0000]          ;LOCATION 6E
U 014F, 0000,00 ;4454          WORD[0000]
                ;4455
U 0150, 0000,00 ;4456          WORD[0000]          ;LOCATION 6F
U 0151, 0000,00 ;4457          WORD[0000]
                ;4458
U 0152, 0000,00 ;4459          WORD[0000]          ;LOCATION 70
U 0153, 0000,00 ;4460          WORD[0000]
                ;4461
U 0154, 0000,00 ;4462          WORD[0000]          ;LOCATION 71
U 0155, 0000,00 ;4463          WORD[0000]
                ;4464
U 0156, 0000,00 ;4465          WORD[0000]          ;LOCATION 72
U 0157, 0000,00 ;4466          WORD[0000]
                ;4467
U 0158, 0000,00 ;4468          WORD[0000]          ;LOCATION 73
U 0159, 0000,00 ;4469          WORD[0000]
                ;4470
U 015A, 0000,00 ;4471          WORD[0000]          ;LOCATION 74
U 015B, 0000,00 ;4472          WORD[0000]
                ;4473
U 015C, 0000,00 ;4474          WORD[0000]          ;LOCATION 75
```

```
U 015D, 0000,00 ;4475           WORD[0000]
                ;4476
U 015E, 0000,00 ;4477           WORD[0000]              ;LOCATION 76
U 015F, 0000,00 ;4478           WORD[0000]
                ;4479
U 0160, 0000,00 ;4480           WORD[0000]              ;LOCATION 77
U 0161, 0000,00 ;4481           WORD[0000]
                ;4482
                ;4483   .TOC    "PROGRAM SPECIFIC DATA SECTION (LS 80-F7)"
                ;4484   ;
                ;4485   ;       This section represents the second half of LS.  It is only acceseble
                ;4486   ;       with a MOV instruction, or one that uses the MOVE microinstruction for-
                ;4487   ;       mat.  This section will be loaded in a seperate transfer.
                ;4488   ;
                ;4489   TRANSFER.DATA.LS2:
U 0162, 0000,78 ;4490           COUNT.LS[0078]          ;LONGWORD COUNT (NUMBER OF LONGWORDS TO TRANS-
                ;4491                                   ; FER TO LS = 120 DECIMAL) DESTINATION IS LS
U 0163, 0000,80 ;4492           START.ADR[0080]         ;DESTINATION START ADDRESS (START AT LS 80(H))
                ;4493
U 0164, 0000,3C ;4494           WORD[003C]              ;LOCATION 80
U 0165, 0000,00 ;4495           WORD[0000]
                ;4496
U 0166, 0000,43 ;4497           WORD[0043]              ;LOCATION 81
U 0167, 0000,00 ;4498           WORD[0000]
                ;4499
U 0168, 0000,64 ;4500           WORD[0064]              ;LOCATION 82
U 0169, 0000,00 ;4501           WORD[0000]
                ;4502
U 016A, 0000,25 ;4503           WORD[0025]              ;LOCATION 83
U 016B, 0000,00 ;4504           WORD[0000]
                ;4505
U 016C, 0000,26 ;4506           WORD[0026]              ;LOCATION 84
U 016D, 0000,00 ;4507           WORD[0000]
                ;4508
U 016E, 0000,20 ;4509           WORD[0020]              ;LOCATION 85
U 016F, 0000,00 ;4510           WORD[0000]
                ;4511
U 0170, 0000,54 ;4512           WORD[0054]              ;LOCATION 86
U 0171, 0000,00 ;4513           WORD[0000]
                ;4514
U 0172, 0000,7D ;4515           WORD[007D]              ;LOCATION 87
U 0173, 0000,00 ;4516           WORD[0000]
                ;4517
U 0174, 00FF,C3 ;4518           WORD[0FFC3]             ;LOCATION 88
U 0175, 00FF,FF ;4519           WORD[0FFFF]
                ;4520
U 0176, 00FF,80 ;4521           WORD[0FF80]             ;LOCATION 89
U 0177, 00FF,FF ;4522           WORD[0FFFF]
                ;4523
U 0178, 0000,00 ;4524           WORD[0000]              ;LOCATION 8A
U 0179, 0000,00 ;4525           WORD[0000]
                ;4526
U 017A, 0000,00 ;4527           WORD[0000]              ;LOCATION 8B
U 017B, 0000,00 ;4528           WORD[0000]
                ;4529
```

```
U 017C, 0000,00 ;4530          WORD[0000]              ;LOCATION 8C
U 017D, 0000,00 ;4531          WORD[0000]
                ;4532
U 017E, 0000,00 ;4533          WORD[0000]              ;LOCATION 8D
U 017F, 0000,00 ;4534          WORD[0000]
                ;4535
U 0180, 0000,00 ;4536          WORD[0000]              ;LOCATION 8E
U 0181, 0000,00 ;4537          WORD[0000]
                ;4538
U 0182, 0000,00 ;4539          WORD[0000]              ;LOCATION 8F
U 0183, 0000,00 ;4540          WORD[0000]
                ;4541
U 0184, 0000,00 ;4542          WORD[0000]              ;LOCATION 90
U 0185, 0000,00 ;4543          WORD[0000]
                ;4544
U 0186, 0000,00 ;4545          WORD[0000]              ;LOCATION 91
U 0187, 0000,00 ;4546          WORD[0000]
                ;4547
U 0188, 0000,00 ;4548          WORD[0000]              ;LOCATION 92
U 0189, 0000,00 ;4549          WORD[0000]
                ;4550
U 018A, 0000,00 ;4551          WORD[0000]              ;LOCATION 93
U 018B, 0000,00 ;4552          WORD[0000]
                ;4553
U 018C, 0000,00 ;4554          WORD[0000]              ;LOCATION 94
U 018D, 0000,00 ;4555          WORD[0000]
                ;4556
U 018E, 0000,00 ;4557          WORD[0000]              ;LOCATION 95
U 018F, 0000,00 ;4558          WORD[0000]
                ;4559
U 0190, 0000,00 ;4560          WORD[0000]              ;LOCATION 96
U 0191, 0000,00 ;4561          WORD[0000]
                ;4562
U 0192, 0000,00 ;4563          WORD[0000]              ;LOCATION 97
U 0193, 0000,00 ;4564          WORD[0000]
                ;4565
U 0194, 0000,00 ;4566          WORD[0000]              ;LOCATION 98
U 0195, 0000,00 ;4567          WORD[0000]
                ;4568
U 0196, 0000,00 ;4569          WORD[0000]              ;LOCATION 99
U 0197, 0000,00 ;4570          WORD[0000]
                ;4571
U 0198, 0000,00 ;4572          WORD[0000]              ;LOCATION 9A
U 0199, 0000,03 ;4573          WORD[0003]
                ;4574
U 019A, 0080,00 ;4575          WORD[8000]              ;LOCATION 9B
U 019B, 007F,FF ;4576          WORD[7FFF]
                ;4577
U 019C, 0000,00 ;4578          WORD[0000]              ;LOCATION 9C
U 019D, 0000,54 ;4579          WORD[0054]
                ;4580
U 019E, 0000,00 ;4581          WORD[0000]              ;LOCATION 9D
U 019F, 0000,F0 ;4582          WORD[00F0]
                ;4583
U 01A0, 0000,00 ;4584          WORD[0000]              ;LOCATION 9E
```

```
U 01A1, 0000,FC ;4585          WORD[00FC]
                ;4586
U 01A2, 003F,FF ;4587          WORD[3FFF]              ;LOCATION 9F
U 01A3, 003F,00 ;4588          WORD[3F00]
                ;4589
U 01A4, 0000,03 ;4590          WORD[0003]              ;LOCATION 0A0
U 01A5, 0000,03 ;4591          WORD[0003]
                ;4592
U 01A6, 0000,00 ;4593          WORD[0000]              ;LOCATION 0A1
U 01A7, 0000,00 ;4594          WORD[0000]
                ;4595
U 01A8, 0000,0F ;4596          WORD[000F]              ;LOCATION 0A2
U 01A9, 0000,0F ;4597          WORD[000F]
                ;4598
U 01AA, 0000,00 ;4599          WORD[0000]              ;LOCATION 0A3
U 01AB, 0000,00 ;4600          WORD[0000]
                ;4601
U 01AC, 0000,EF ;4602          WORD[00EF]              ;LOCATION 0A4
U 01AD, 0000,EF ;4603          WORD[00EF]
                ;4604
U 01AE, 0000,00 ;4605          WORD[0000]              ;LOCATION 0A5
U 01AF, 0000,00 ;4606          WORD[0000]
                ;4607
U 01B0, 000F,EF ;4608          WORD[0FEF]              ;LOCATION 0A6
U 01B1, 000F,EF ;4609          WORD[0FEF]
                ;4610
U 01B2, 0000,00 ;4611          WORD[0000]              ;LOCATION 0A7
U 01B3, 0000,00 ;4612          WORD[0000]
                ;4613
U 01B4, 0088,8B ;4614          WORD[888B]              ;LOCATION 0A8
U 01B5, 0088,8B ;4615          WORD[888B]
                ;4616
U 01B6, 0077,74 ;4617          WORD[7774]              ;LOCATION 0A9
U 01B7, 0000,00 ;4618          WORD[0000]
                ;4619
U 01B8, 00CC,CF ;4620          WORD[0CCCF]             ;LOCATION 0AA
U 01B9, 00CC,CF ;4621          WORD[0CCCF]
                ;4622
U 01BA, 0033,30 ;4623          WORD[3330]              ;LOCATION 0AB
U 01BB, 0000,00 ;4624          WORD[0000]
                ;4625
U 01BC, 00EE,EF ;4626          WORD[0EEEF]             ;LOCATION 0AC
U 01BD, 00EE,EF ;4627          WORD[0EEEF]
                ;4628
U 01BE, 0011,10 ;4629          WORD[1110]              ;LOCATION 0AD
U 01BF, 0000,00 ;4630          WORD[0000]
                ;4631
U 01C0, 00FF,EF ;4632          WORD[0FFEF]             ;LOCATION 0AE
U 01C1, 00FF,EF ;4633          WORD[0FFEF]
                ;4634
U 01C2, 0000,10 ;4635          WORD[0010]              ;LOCATION 0AF
U 01C3, 0000,00 ;4636          WORD[0000]
                ;4637
U 01C4, 0000,00 ;4638          WORD[0000]              ;LOCATION B0
U 01C5, 0000,00 ;4639          WORD[0000]
```

```
                  ;4640
U 01C6, 0000,00   ;4641          WORD[0000]          ;LOCATION B1
U 01C7, 0000,00   ;4642          WORD[0000]
                  ;4643
U 01C8, 0000,00   ;4644          WORD[0000]          ;LOCATION B2
U 01C9, 0000,00   ;4645          WORD[0000]
                  ;4646
U 01CA, 0000,00   ;4647          WORD[0000]          ;LOCATION B3
U 01CB, 0000,00   ;4648          WORD[0000]
                  ;4649
U 01CC, 0000,00   ;4650          WORD[0000]          ;LOCATION B4
U 01CD, 0000,00   ;4651          WORD[0000]
                  ;4652
U 01CE, 0000,00   ;4653          WORD[0000]          ;LOCATION B5
U 01CF, 0000,00   ;4654          WORD[0000]
                  ;4655
U 01D0, 0000,00   ;4656          WORD[0000]          ;LOCATION 0B6
U 01D1, 0000,00   ;4657          WORD[0000]
                  ;4658
U 01D2, 0000,00   ;4659          WORD[0000]          ;LOCATION 0B7
U 01D3, 0000,00   ;4660          WORD[0000]
                  ;4661
U 01D4, 0000,00   ;4662          WORD[0000]          ;LOCATION 0B8
U 01D5, 0000,00   ;4663          WORD[0000]
                  ;4664
U 01D6, 0000,00   ;4665          WORD[0000]          ;LOCATION 0B9
U 01D7, 0000,00   ;4666          WORD[0000]
                  ;4667
U 01D8, 0000,00   ;4668          WORD[0000]          ;LOCATION 0BA
U 01D9, 0000,00   ;4669          WORD[0000]
                  ;4670
U 01DA, 0000,00   ;4671          WORD[0000]          ;LOCATION 0BB
U 01DB, 0000,00   ;4672          WORD[0000]
                  ;4673
U 01DC, 0000,00   ;4674          WORD[0000]          ;LOCATION 0BC
U 01DD, 0000,00   ;4675          WORD[0000]
                  ;4676
U 01DE, 0000,00   ;4677          WORD[0000]          ;LOCATION 0BD
U 01DF, 0000,00   ;4678          WORD[0000]
                  ;4679
U 01E0, 0000,00   ;4680          WORD[0000]          ;LOCATION 0BE
U 01E1, 0000,00   ;4681          WORD[0000]
                  ;4682
U 01E2, 0000,00   ;4683          WORD[0000]          ;LOCATION 0BF
U 01E3, 0000,00   ;4684          WORD[0000]
                  ;4685
U 01E4, 0000,00   ;4686          WORD[0000]          ;LOCATION 0C0
U 01E5, 0000,00   ;4687          WORD[0000]
                  ;4688
U 01E6, 0000,00   ;4689          WORD[0000]          ;LOCATION 0C1
U 01E7, 0000,00   ;4690          WORD[0000]
                  ;4691
U 01E8, 0000,00   ;4692          WORD[0000]          ;LOCATION 0C2
U 01E9, 0000,00   ;4693          WORD[0000]
                  ;4694
```

```
U 01EA, 0000,00 ;4695          WORD[0000]              ;LOCATION 0C3
U 01EB, 0000,00 ;4696          WORD[0000]
                ;4697
U 01EC, 0000,00 ;4698          WORD[0000]              ;LOCATION 0C4
U 01ED, 0000,00 ;4699          WORD[0000]
                ;4700
U 01EE, 0000,00 ;4701          WORD[0000]              ;LOCATION 0C5
U 01EF, 0000,00 ;4702          WORD[0000]
                ;4703
U 01F0, 0000,00 ;4704          WORD[0000]              ;LOCATION 0C6
U 01F1, 0000,00 ;4705          WORD[0000]
                ;4706
U 01F2, 0000,00 ;4707          WORD[0000]              ;LOCATION 0C7
U 01F3, 0000,00 ;4708          WORD[0000]
                ;4709
U 01F4, 0000,00 ;4710          WORD[0000]              ;LOCATION 0C8
U 01F5, 0000,00 ;4711          WORD[0000]
                ;4712
U 01F6, 0000,00 ;4713          WORD[0000]              ;LOCATION 0C9
U 01F7, 0000,00 ;4714          WORD[0000]
                ;4715
U 01F8, 0000,00 ;4716          WORD[0000]              ;LOCATION 0CA
U 01F9, 0000,00 ;4717          WORD[0000]
                ;4718
U 01FA, 0000,00 ;4719          WORD[0000]              ;LOCATION 0CB
U 01FB, 0000,00 ;4720          WORD[0000]
                ;4721
U 01FC, 0000,00 ;4722          WORD[0000]              ;LOCATION 0CC
U 01FD, 0000,00 ;4723          WORD[0000]
                ;4724
U 01FE, 0000,00 ;4725          WORD[0000]              ;LOCATION 0CD
U 01FF, 0000,00 ;4726          WORD[0000]
                ;4727
U 0200, 0000,00 ;4728          WORD[0000]              ;LOCATION 0CE
U 0201, 0000,00 ;4729          WORD[0000]
                ;4730
U 0202, 0000,00 ;4731          WORD[0000]              ;LOCATION 0CF
U 0203, 0000,00 ;4732          WORD[0000]
                ;4733
U 0204, 0000,00 ;4734          WORD[0000]              ;LOCATION 0D0
U 0205, 0000,00 ;4735          WORD[0000]
                ;4736
U 0206, 0000,00 ;4737          WORD[0000]              ;LOCATION 0D1
U 0207, 0000,00 ;4738          WORD[0000]
                ;4739
U 0208, 0000,00 ;4740          WORD[0000]              ;LOCATION 0D2
U 0209, 0000,00 ;4741          WORD[0000]
                ;4742
U 020A, 0000,00 ;4743          WORD[0000]              ;LOCATION 0D3
U 020B, 0000,00 ;4744          WORD[0000]
                ;4745
U 020C, 0000,00 ;4746          WORD[0000]              ;LOCATION 0D4
U 020D, 0000,00 ;4747          WORD[0000]
                ;4748
U 020E, 0000,00 ;4749          WORD[0000]              ;LOCATION 0D5
```

```
U 020F, 0000,00 ;4750           WORD[0000]
                ;4751
U 0210, 0000,00 ;4752           WORD[0000]              ;LOCATION 0D6
U 0211, 0000,00 ;4753           WORD[0000]
                ;4754
U 0212, 0000,00 ;4755           WORD[0000]              ;LOCATION 0D7
U 0213, 0000,00 ;4756           WORD[0000]
                ;4757
U 0214, 0000,00 ;4758           WORD[0000]              ;LOCATION 0D8
U 0215, 0000,00 ;4759           WORD[0000]
                ;4760
U 0216, 0000,00 ;4761           WORD[0000]              ;LOCATION 0D9
U 0217, 0000,00 ;4762           WORD[0000]
                ;4763
U 0218, 0000,00 ;4764           WORD[0000]              ;LOCATION 0DA
U 0219, 0000,00 ;4765           WORD[0000]
                ;4766
U 021A, 0000,00 ;4767           WORD[0000]              ;LOCATION 0DB
U 021B, 0000,00 ;4768           WORD[0000]
                ;4769
U 021C, 0000,00 ;4770           WORD[0000]              ;LOCATION 0DC
U 021D, 0000,00 ;4771           WORD[0000]
                ;4772
U 021E, 0000,00 ;4773           WORD[0000]              ;LOCATION 0DD
U 021F, 0000,00 ;4774           WORD[0000]
                ;4775
U 0220, 0000,00 ;4776           WORD[0000]              ;LOCATION 0DE
U 0221, 0000,00 ;4777           WORD[0000]
                ;4778
U 0222, 0000,00 ;4779           WORD[0000]              ;LOCATION 0DF
U 0223, 0000,00 ;4780           WORD[0000]
                ;4781
U 0224, 0000,00 ;4782           WORD[0000]              ;LOCATION 0E0
U 0225, 0000,00 ;4783           WORD[0000]
                ;4784
U 0226, 0000,00 ;4785           WORD[0000]              ;LOCATION 0E1
U 0227, 0000,00 ;4786           WORD[0000]
                ;4787
U 0228, 0000,00 ;4788           WORD[0000]              ;LOCATION 0E2
U 0229, 0000,00 ;4789           WORD[0000]
                ;4790
U 022A, 0000,00 ;4791           WORD[0000]              ;LOCATION 0E3
U 022B, 0000,00 ;4792           WORD[0000]
                ;4793
U 022C, 0000,00 ;4794           WORD[0000]              ;LOCATION 0E4
U 022D, 0000,00 ;4795           WORD[0000]
                ;4796
U 022E, 0000,00 ;4797           WORD[0000]              ;LOCATION 0E5
U 022F, 0000,00 ;4798           WORD[0000]
                ;4799
U 0230, 0000,00 ;4800           WORD[0000]              ;LOCATION 0E6
U 0231, 0000,00 ;4801           WORD[0000]
                ;4802
U 0232, 0000,00 ;4803           WORD[0000]              ;LOCATION 0E7
U 0233, 0000,00 ;4804           WORD[0000]
```

```
                    ;4805
U 0234, 0000,00     ;4806           WORD[0000]              ;LOCATION 0E8
U 0235, 0000,00     ;4807           WORD[0000]
                    ;4808
U 0236, 0000,00     ;4809           WORD[0000]              ;LOCATION 0E9
U 0237, 0000,00     ;4810           WORD[0000]
                    ;4811
U 0238, 0000,00     ;4812           WORD[0000]              ;LOCATION 0EA
U 0239, 0000,00     ;4813           WORD[0000]
                    ;4814
U 023A, 0000,00     ;4815           WORD[0000]              ;LOCATION 0EB
U 023B, 0000,00     ;4816           WORD[0000]
                    ;4817
U 023C, 0000,00     ;4818           WORD[0000]              ;LOCATION 0EC
U 023D, 0000,00     ;4819           WORD[0000]
                    ;4820
U 023E, 0000,00     ;4821           WORD[0000]              ;LOCATION 0ED
U 023F, 0000,00     ;4822           WORD[0000]
                    ;4823
U 0240, 0000,00     ;4824           WORD[0000]              ;LOCATION 0EE
U 0241, 0000,00     ;4825           WORD[0000]
                    ;4826
U 0242, 0000,00     ;4827           WORD[0000]              ;LOCATION 0EF
U 0243, 0000,00     ;4828           WORD[0000]
                    ;4829
U 0244, 0000,00     ;4830           WORD[0000]              ;LOCATION 0F0
U 0245, 0000,00     ;4831           WORD[0000]
                    ;4832
U 0246, 0000,00     ;4833           WORD[0000]              ;LOCATION 0F1
U 0247, 0000,00     ;4834           WORD[0000]
                    ;4835
U 0248, 0000,00     ;4836           WORD[0000]              ;LOCATION 0F2
U 0249, 0000,00     ;4837           WORD[0000]
                    ;4838
U 024A, 0000,00     ;4839           WORD[0000]              ;LOCATION 0F3
U 024B, 0000,00     ;4840           WORD[0000]
                    ;4841
U 024C, 0000,00     ;4842           WORD[0000]              ;LOCATION 0F4
U 024D, 0000,00     ;4843           WORD[0000]
                    ;4844
U 024E, 0000,00     ;4845           WORD[0000]              ;LOCATION 0F5
U 024F, 0000,00     ;4846           WORD[0000]
                    ;4847
U 0250, 0000,00     ;4848           WORD[0000]              ;LOCATION 0F6
U 0251, 0000,00     ;4849           WORD[0000]
                    ;4850
U 0252, 0000,00     ;4851           WORD[0000]              ;LOCATION 0F7
U 0253, 0000,00     ;4852           WORD[0000]
                    ;4853
                    ;4854                                   ;IF ANY TRANSFER DATA IS ADDED, PUT BEFORE THIS
                    ;4855                                   ; LINE.  LIMIT OF 962 ADDITIONAL WORDS (470
                    ;4856                                   ; LONGWORD TRANSFERS).
                    ;4857   .REGION/600,6FF
```

```
                ;4858   .PAGE   "WCS SUBROUTINES FOR CONSOLE USE"
                ;4859   .TOC    "  GENERATE TRANSFER DATA"
                ;4860   ;
                ;4861   ;       This routine is used by the diagnostic monitor to load WR 0 with a
                ;4862   ;       data pattern from the console.  The monitor will set CONSOLE ATTN if
                ;4863   ;       a 1 is to be generated, or clear CONSOLE ATTN if a 0 is to be gen-
                ;4864   ;       erated.  It will then single step this routine to shift the data bit
                ;4865   ;       into WR 0.  This routine loads a 32 bit value much more quickly than
                ;4866   ;       shifting each instruction into the CSR from the monitor, and is used
                ;4867   ;       mainly to set up data to load in LS for constants.
                ;4868   ;
                ;4869
                ;4870   ;                               ***WARNING***
                ;4871   ;
                ;4872   ;       This routine must start at 600 and end at 605.  DO NOT move this
                ;4873   ;       routine to any other area!!
                ;4874   ;
                ;4875   ;
                ;4876
                ;4877   GEN.XFER.DATA:
U 0600, 2F80,15 ;4878           CLR WR[0]                       ;CLEAR A WORKING REG
U 0601, A0C0,15 ;4879           COM WR[0]                       ;NOW WR 0 IS ALL ONES
                ;4880   LOOP.XFER:
U 0602, 5B00,18 ;4881           SKIP.IF[CONSOLE.ATTN]           ;SKIP NEXT INSTRUCTION IS CONSOLE ATTN
                ;4882                                           ; IS SET (GENERATING A 1)
                ;4883           ASHL WR[0],                     ;SHIFT A 0 INTO BIT 0 OF WR 0
U 0603, A3D0,1E ;4884            SKIP                           ; AND SKIP ROL
U 0604, A3C0,15 ;4885           ROL WR[0]                       ;SHIFT A 1 INTO BIT 0 OF WR 0
U 0605, 0860,24 ;4886           JMP [LOOP.XFER]                 ;REPEAT
                ;4887
                ;4888
                ;4889   ;
                ;4890   ;       DIAGNOSTIC VERSION NUMBER IS STORED HERE
                ;4891   ;
                ;4892
                ;4893   ;                               ***WARNING***
                ;4894   ;
                ;4895   ;       These words must be at location 606 and 607.  DO NOT move this word
                ;4896   ;       to any other area!!
                ;4897   ;
                ;4898   ;
                ;4899
U 0606, 0001,20 ;4900           WORD[0120]                      ;VERSION 01.20
U 0607, 0043,43 ;4901           ASCII [CC]                      ;LAST 2 LETTERS OF FILE NAME [ENKCC]
                ;4902
                ;4903
```

```
;4904   .PAGE   '' DEPOSIT MCT CSR ROUTINE''
;4905   ;+++
;4906   ;
;4907   ; FUNCTIONAL DESCRIPTION:
;4908   ;
;4909   ;       The deposit to CSR routine is used to write the memory controller's
;4910   ;       control and status register.  The memory controller has three CSRs
;4911   ;       named CSR 0, CSR 1, and CSR 2.
;4912   ;         CSR 0 contains checkbits or syndrome bits returned from the ECC logic
;4913   ;       after certain diagnostic routines.  It is NOT writable directly with
;4914   ;       this routine.
;4915   ;         CSR 1 contains error bits set if an error occurs on a memory access.
;4916   ;       It does contain five maintenance bits (bits 29-25) which are writable.
;4917   ;       There are also seven checkbits (bits 6-0) which are writable, but not
;4918   ;       readable.  These bits are used to write checkbits into the ECC chips.
;4919   ;         CSR 2 contains error bits set if a UNIBUS error occurs on a UNIBUS
;4920   ;       access.  These are read only bits and are NOT writable by this routine.
;4921   ;         Therefore, CSR 1 is the only writable CSR, and only bits 29-25 can be
;4922   ;       both written and read back.  This routine will thus force CSR 1 on a
;4923   ;       deposit to CSR.
;4924   ;         This routine will write LS 5 with data to access CSR 1, then write
;4925   ;       the data residing in LS 6 into the CSR.  It will then issue CPU ATTN
;4926   ;       to inform the console that it has completed the function.
;4927   ;         NOTE:  The error bits of CSR 1 are cleared on any write to CSR 1.
;4928   ;       These are bits 31, 30 and 23-14.Bits 13-0 and bit 24 are unused.
;4929   ;
;4930   ; CALLING SEQUENCE:
;4931   ;
;4932   ;       The console loads the address of a pointer to this routine (a JMP in-
;4933   ;       struction at WCS location 50) into the UPC and starts the CPU.
;4934   ;
;4935   ; INPUT PARAMETERS:
;4936   ;
;4937   ;       LS location 6 must have been written by the console with the desired
;4938   ;       data to be deposited in CSR 1 previous to executing this routine.
;4939   ;
;4940   ; IMPLICIT INPUTS:
;4941   ;
;4942   ;       None
;4943   ;
;4944   ; OUTPUT PARAMATERS:
;4945   ;
;4946   ;       CSR 1 bits 29-25 and 6-0 are written with data from LS 6.
;4947   ;
;4948   ; IMPLICIT OUTPUTS:
;4949   ;
;4950   ;       None                                                          U 0
;4951   ;
;4952   ; SIDE EFFECTS:
;4953   ;
;4954   ;       WR 0 will be modified by this routine.
;4955   ;       LS 5 will be modified by this routine.                        U 0
;4956   ;       CSR 1 bits 31, 30 and 23-14 are cleared.
;4957   ;                                                                     U 0
;4958   ;---
```

```
                    ;4959
                    ;4960   DEPOSIT.CSR:
                    ;4961
U 0608, 3644,15     ;4962           MOV LS[CSR1] TO WR[0]           ;SET BIT 2 IN WR 0 AS CSR NUMBER TO
                    ;4963                                           ; WRITE (BITS 2 AND 3 ARE CSR NUMBER)
U 0609, BE0A,15     ;4964           MOV WR[0] TO LS[T5.SUB]         ;PUT CSR NUMBER IN LS LOCATION 5 (CSR
                    ;4965                                           ; 1)
U 060A, 1D0B,F5     ;4966           MEM.REQ[WRITE.CSR] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO
                    ;4967                                           ; ACCESS CSR 1
U 060B, B20C,15     ;4968           WRITE.MEM LS[T6.SUB]            ;SEND DATA IN LS LOCATION 6 TO CSR 1
U 060C, 8868,74     ;4969           JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
```

```
                ;4970   .PAGE   '' EXAMINE MCT CSR ROUTINE''
                ;4971   ;+++
                ;4972   ;
                ;4973   ; FUNCTIONAL DESCRIPTION:
                ;4974   ;
                ;4975   ;       The examine CSR routine reads the memory controller's control and
                ;4976   ;       status register.  There are 3 CSRs labeled CSR 0, CSR 1 and CSR 2.
                ;4977   ;         CSR 0 contains checkbits or syndrome bits from the ECC logic.
                ;4978   ;       These are contained in bits 6-0 of the CSR.  Other bits are UNPRE-
                ;4979   ;       DICTABLE.  Bits 6-0 are only written by special diagnostic routines
                ;4980   ;       (they are not writtable by the DEPOSIT CSR command).
                ;4981   ;         CSR 1 contains error bits and maintenance bits.  The error bits are
                ;4982   ;       31, 30, and 23-14.  Maintenance bits are 29-25.  The other bits are
                ;4983   ;       UNPREDICTABLE.  The error bits are written during memory functions
                ;4984   ;       only and are not writable via the DEPOSIT CSR command. The maintenance
                ;4985   ;       bits are writable.  Note that a DEPOSIT CSR command will write the
                ;4986   ;       maintenance bits and clear all error bits.
                ;4987   ;         CSR 2 contains three bits (bits 16-14) which are readable.  These are
                ;4988   ;       set when a UNIBUS error occurs on a UNIBUS access.  They are not
                ;4989   ;       directly writable with the DEPOSIT CSR command.
                ;4990   ;         The console loads LS 5 with the CSR number (0, 1, or 2) to be read.
                ;4991   ;       This routine rotates LS 5 two places left to position it correctly.
                ;4992   ;       It then executes a read CSR and places the data in LS 6 for the console
                ;4993   ;       to read.
                ;4994   ;
                ;4995   ; CALLING SEQUENCE:
                ;4996   ;
                ;4997   ;       The console loads the address of a pointer to this routine (a JMP in-
                ;4998   ;       struction at WCS location 51) into the UPC and starts the CPU.
                ;4999   ;
                ;5000   ; INPUT PARAMETERS:
                ;5001   ;
                ;5002   ;       LS 5 contains the CSR number to be read.
                ;5003   ;
                ;5004   ; IMPLICIT INPUTS:
                ;5005   ;
                ;5006   ;       None
                ;5007   ;
                ;5008   ; OUTPUT PARAMATERS:
                ;5009   ;
                ;5010   ;       LS 6 - Data read from CSR selected.
                ;5011   ;
                ;5012   ; IMPLICIT OUTPUTS:
                ;5013   ;
                ;5014   ;       None
                ;5015   ;
                ;5016   ; SIDE EFFECTS:
                ;5017   ;
                ;5018   ;       LS 5 is rotated 2 places left.
                ;5019   ;
                ;5020   ;---
                ;5021
                ;5022   EXAMINE.CSR:
                ;5023
U 060D, 360A,15 ;5024           MOV LS[T5.SUB] TO WR[0]         ;GET CSR NUMBER FROM LS 5
```

```
U 060E, A2D0,15 ;5025          SHL2 WR[0]                          ;SHIFT NUMBER 2 PLACES LEFT TO GET
                ;5026                                              ; CSR NUMBER IN BITS 2 AND 3
U 060F, BE0A,15 ;5027          MOV WR[0] TO LS[T5.SUB]             ;PUT SHIFTED NUMBER IN LS 5
                ;5028
U 0610, 9D0B,75 ;5029          MEM.REQ[READ.CSR] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO
                ;5030                                              ; ACCESS CSR SPECIFIED
U 0611, 3022,15 ;5031          MOV MEM.DATA TO WR[0]               ;READ CSR SELECTED
                ;5032
                ;5033  CHECK.CSR2:
U 0612, B60A,95 ;5034          MOV LS[T5.SUB] TO WR[1]             ;GET SHIFTED CSR NUMBER
                ;5035          CMP LS[#8] WITH WR[1],              ;SEE IF CSR 2 WAS SELECTED
U 0613, CE46,B5 ;5036           DT(LONG)&SET.ALU.CC                ; SET UP CONDITION CODES
U 0614, 0861,61 ;5037          JMP.IF[NEQ] TO [CHECK.CSR1]         ;JUMP IF CSR 2 WAS NOT SELECTED
U 0615, C530,15 ;5038          BIC LS[CSR2.MASK] TO WR[0]          ;CLEAR UNUSED (UNPREDICTABLE) BITS IN
                ;5039                                              ; CSR 2.
                ;5040  CHECK.CSR1:
                ;5041          CMP LS[#4] WITH WR[1],              ;SEE IF CSR 1 WAS SELECTED
U 0616, 4E44,B5 ;5042           DT(LONG)&SET.ALU.CC                ; SET UP CONDITION CODES
U 0617, 0861,91 ;5043          JMP.IF[NEQ] TO [CHECK.CSR0]         ;JUMP IF CSR 1 WAS NOT SELECTED
U 0618, C52E,15 ;5044          BIC LS[CSR1.MASK] TO WR[0]          ;CLEAR UNUSED (UNPREDICTABLE) BITS IN
                ;5045                                              ; CSR 1.
                ;5046  CHECK.CSR0:
                ;5047          CMP LS[#0] WITH WR[1],              ;SEE IF CSR 0 WAS SELECTED
U 0619, 4E9C,B5 ;5048           DT(LONG)&SET.ALU.CC                ; SET UP CONDITION CODES
U 061A, 8861,D1 ;5049          JMP.IF[NEQ] TO [LOAD.LS6]           ;JUMP IF CSR 0 WAS NOT SELECTED
U 061B, 452C,15 ;5050          BIC LS[CSR0.MASK] TO WR[0]          ;CLEAR UNUSED (UNPREDICTABLE) BITS IN
                ;5051                                              ; CSR 0.
U 061C, C54E,15 ;5052          BIC LS[BIT7] TO WR[0]               ;BIT 7 IS ALSO UNUSED
                ;5053
                ;5054  LOAD.LS6:
U 061D, BE0C,15 ;5055          MOV WR[0] TO LS[T6.SUB]             ;LOAD CSR DATA INTO LS 6 FOR PRINTOUT
U 061E, 8868,74 ;5056          JMP [WAIT.SUB]                      ;ISSUE CPU ATTN AND WAIT FOR RESPONSE
```

```
                ;5057   .page   "  DEPOSIT MCT TRANSLATION BUFFER ROUTINE"
                ;5058   ;+++
                ;5059   ;
                ;5060   ; FUNCTIONAL DESCRIPTION:
                ;5061   ;
                ;5062   ;       This routine allows a write to the translation buffer portion of the
                ;5063   ;       memory controller.  The buffer area contains 128 decimal locations
                ;5064   ;       addressed from 0-7F(H).  The remainder of the TB RAM is either unused
                ;5065   ;       or contains the UNIBUS MAP.  The DEPOSIT UB command is used to write
                ;5066   ;       the UNIBUS map portion of the TB RAMs.  The address specified with the
                ;5067   ;       DEPOSIT command will be the actual RAM address accessed.  This routine
                ;5068   ;       will do any shifting necessary on the address specified to position it
                ;5069   ;       correctly.  The address specified has been loaded by the console in LS
                ;5070   ;       location 5 prior to executing this routine.  The address must be in HEX
                ;5071   ;       format.
                ;5072   ;         The data specified has been placed in LS 6 by the console prior to
                ;5073   ;       executing this routine.  Bits 07-01 will be the Valid, Prot, Modify,
                ;5074   ;       and Byte Offset bits (there are four Prot bits).  Bits 22-08 will
                ;5075   ;       contain the PA bits from PA 23 through PA 09 respectively.  Bits 31
                ;5076   ;       through 23 and bit 0 are unused. This routine will do any shifting nec-
                ;5077   ;       essary to write the bits in TB as described.  The data specified must
                ;5078   ;       be in HEX format.
                ;5079   ;
                ;5080   ; CALLING SEQUENCE:
                ;5081   ;
                ;5082   ;       The console loads the address of a pointer to this routine (a JMP in-
                ;5083   ;       struction at WCS location 52) into the UPC and starts the CPU.
                ;5084   ;
                ;5085   ; INPUT PARAMETERS:
                ;5086   ;
                ;5087   ;       LS 5 - TB address to be written (range 0-7F)
                ;5088   ;       LS 6 - TB data to write (22 bits, from 1-22).  Bits 0 and 23-31 are
                ;5089   ;              ignored.
                ;5090   ;
                ;5091   ; IMPLICIT INPUTS:
                ;5092   ;
                ;5093   ;       None
                ;5094   ;
                ;5095   ; OUTPUT PARAMATERS:
                ;5096   ;
                ;5097   ;       TB will be written with specified data at specified address
                ;5098   ;
                ;5099   ; IMPLICIT OUTPUTS:
                ;5100   ;
                ;5101   ;       None
                ;5102   ;
                ;5103   ; SIDE EFFECTS:
                ;5104   ;
                ;5105   ;       WR 0 will be modified
                ;5106   ;       WR 1 will be modified
                ;5107   ;
                ;5108   ;---
                ;5109
                ;5110   DEPOSIT.TB:
U 061F, 8862,FC ;5111          JSR [SHIFT.TB.ADR]              ;GO TO SUBROUTINE TO POSITION TB AD-
```

```
                ;5112                                   ; DRESS CORRECTLY
U 0620, 360C,15 ;5113           MOV LS[T6.SUB] TO WR[0] ;GET DATA FROM LS 6
U 0621, 4526,15 ;5114           BIC LS[#FF] TO WR[0]    ;CLEAR LOW BYTE OF DATA SPECIFIED.
                ;5115                                   ; LOW BYTE WILL BE PUT IN BITS 24-31
                ;5116                                   ; LATER
U 0622, 8862,AC ;5117           JSR [SHIFT.LOOP]        ;GO TO SUBROUTINE TO SHIFT WR 0 EIGHT
                ;5118                                   ; PLACES RIGHT
U 0623, E40C,15 ;5119           SWAP LS[T6.SUB] WITH WR[0] ;SWAP ORIGINAL DATA SPECIFIED WITH MOD-
                ;5120                                   ; IFIED DATA.  NOW LS 6 CONTAINS BITS
                ;5121                                   ; 8-22 IN BITS 0-14 AS REQUIRED.  WR 0
                ;5122                                   ; CONTAINS ORIGINAL DATA SPECIFIED
U 0624, 452C,15 ;5123           BIC LS[#FFFFFF00] TO WR[0] ;CLEAR ALL BUT LOW BYTE OF ORIGINAL
                ;5124                                   ; DATA SPECIFIED
U 0625, 8862,AC ;5125           JSR [SHIFT.LOOP]        ;GO TO SUBROUTINE TO SHIFT WR 0 EIGHT
                ;5126                                   ; PLACES RIGHT.  ON RETURN, WR 0 WILL
                ;5127                                   ; HAVE BITS 0-7 IN BITS 24-31, OTHER
                ;5128                                   ; BITS ARE CLEAR
U 0626, ED0C,15 ;5129           BIS WR[0] TO LS[T6.SUB] ;NOW LS 6 CONTAINS DATA TO WRITE IN
                ;5130                                   ; TB IN CORRECT FORMAT  I.E. PA BITS
                ;5131                                   ; IN BITS 00-14 AND BYTE OFFSET, MODIFY
                ;5132                                   ; PROT A THROUGH PROT D, AND VALID IN
                ;5133                                   ; BITS 25-31.
                ;5134
U 0627, 980A,F5 ;5135           MEM.REQ[WRITE.TB] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO WRITE
                ;5136                                   ; THE TB
U 0628, B20C,15 ;5137           WRITE.MEM LS[T6.SUB]    ;WRITE DATA FROM LS 6 IN TB
U 0629, 8868,74 ;5138           JMP [WAIT.SUB]          ;JUMP TO SET ATTN AND WAIT
                ;5139
                ;5140   ;
                ;5141   ;       SUBROUTINE TO SHIFT WR 0 TO THE RIGHT 8 PLACES
                ;5142   ;
                ;5143   SHIFT.LOOP:
U 062A, 3646,95 ;5144           MOV LS[#8] TO WR[1]     ;COUNTER IN WR 1 TO SHIFT 8 PLACES
                ;5145   SHIFT.LOOP.1:
U 062B, 2340,15 ;5146           ROR WR[0]               ;SHIFT WR 0 ONE PLACE RIGHT
                ;5147           DEC WR[1],              ;DECREMENT COUNTER
U 062C, A102,B5 ;5148            DT(LONG)&SET.ALU.CC    ; AND SET CONDITION CODES
U 062D, 8862,B1 ;5149           JMP.IF[NEQ] TO [SHIFT.LOOP.1] ;REPEAT UNTIL 8 SHIFTS HAVE OCCURRED
U 062E, 5B00,14 ;5150           RETURN                  ;THEN RETURN
                ;5151   ;
                ;5152   ;       SUBROUTINE TO POSITION TB ADDRESS IN BITS 9-14 AND BIT 31 (TB ADDRESS
                ;5153   ;       BIT 0 MUST BE IN BIT 31).
                ;5154   ;
                ;5155   SHIFT.TB.ADR:
U 062F, 360A,15 ;5156           MOV LS[T5.SUB] TO WR[0] ;GET TB ADDRESS FROM LS 5
U 0630, 452C,15 ;5157           BIC LS[#FFFFFF00] TO WR[0] ;CLEAR ALL BUT LOW BYTE (8 BIT ADDRESS)
U 0631, A2D0,15 ;5158           SHL2 WR[0]              ;SHIFT 2 PLACES LEFT TO POSITION ADDRESS
U 0632, A2D0,15 ;5159           SHL2 WR[0]              ;SHIFT 2 PLACES LEFT TO POSITION ADDRESS
U 0633, A2D0,15 ;5160           SHL2 WR[0]              ;SHIFT 2 PLACES LEFT TO POSITION ADDRESS
U 0634, A2D0,15 ;5161           SHL2 WR[0]              ;SHIFT 2 PLACES LEFT TO POSITION ADDRESS
U 0635, 23D0,15 ;5162           ASHL WR[0]              ;SHIFT ONE MORE PLACE
                ;5163                                   ; NOW BITS 0-6 IN BITS 9-15
                ;5164           BIT LS[BIT15] WITH WR[0], ;SEE IF BIT 15 (MSB) IS SET OR CLEAR.
U 0636, 595E,35 ;5165            DT(LONG)&SET.ALU.CC    ; MSB MUST GO IN BIT 31 IN ORDER TO AD-
                ;5166                                   ; DRESS TB CORRECTLY
```

```
U 0637, 5B00,09 ;5167          SKIP.IF[EQL]                    ;SKIP NEXT INSTRUCTION IF MSB IS 0
U 0638, 477E,15 ;5168          BIS LS[BIT31] TO WR[0]          ;ELSE SET BIT 31 FOR MSB
                ;5169
                ;5170          MOV WR[0] TO LS[T5.SUB],        ;NOW TB ADDRESS IS IN CORRECT FORM IN
                ;5171                                          ; LS LOCATION 5
U 0639, 3E0A,14 ;5172           RETURN                         ; THEN RETURN TO MAIN CODE
```

```
                ;5173   .PAGE   "  EXAMINE TRANSLATION BUFFER ROUTINE"
                ;5174   ;+++
                ;5175   ;
                ;5176   ; FUNCTIONAL DESCRIPTION:
                ;5177   ;
                ;5178   ;       This routine will read the translation buffer of the memory controller.
                ;5179   ;       The TB will be returned with the byte offset, modify, prot A through
                ;5180   ;       prot D, and valid in bits 01-07 respectively.  PA bits 09-23 are re-
                ;5181   ;       turned in bits 08-22 respectively.  Bits 23-31 and bit 0 are not used
                ;5182   ;       and so will be cleared before printing the result.  The result will be
                ;5183   ;       put in LS 6 for the console to print.
                ;5184   ;         The routine will issue a memory request with memory function=READ.TB
                ;5185   ;       and data type=LONGWORD.  The console will load LS 5 with the address
                ;5186   ;       specified before executing this routine.  The routine will shift the
                ;5187   ;       address around as required to address the TB location specified.  The
                ;5188   ;       address specified must be in HEX format.
                ;5189   ;         The TB consists of 128 decimal locations (addresses from 0-7F).  The
                ;5190   ;       remaining locations in the TB RAM are either unused or used by the
                ;5191   ;       UNIBUS map.  UNIBUS map addresses are read via the EXAMINE UB routine.
                ;5192   ;
                ;5193   ; CALLING SEQUENCE:
                ;5194   ;
                ;5195   ;       The console loads the address of a pointer to this routine (a JMP in-
                ;5196   ;       struction at WCS location 53) into the UPC and starts the CPU.
                ;5197   ;
                ;5198   ; INPUT PARAMETERS:
                ;5199   ;
                ;5200   ;       LS 5 - TB address (0-7F) in hex format
                ;5201   ;
                ;5202   ; IMPLICIT INPUTS:
                ;5203   ;
                ;5204   ;       None
                ;5205   ;
                ;5206   ; OUTPUT PARAMATERS:
                ;5207   ;
                ;5208   ;       LS 6 - TB data from address specified
                ;5209   ;
                ;5210   ; IMPLICIT OUTPUTS:
                ;5211   ;
                ;5212   ;       None
                ;5213   ;
                ;5214   ; SIDE EFFECTS:
                ;5215   ;
                ;5216   ;       LS 5 is modified
                ;5217   ;       WR 0 is modified
                ;5218   ;
                ;5219   ;---
                ;5220
                ;5221   EXAMINE.TB:
U 063A, 8862,FC ;5222           JSR [SHIFT.TB.ADR]              ;SHIFT ADDRESS SPECIFIED FROM LS 5 INTO
                ;5223                                           ; CORRECT POSITION AND REPLACE IN LS 5
U 063B, 9C0B,F5 ;5224           MEM.REQ[READ.TB] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO READ
                ;5225                                           ; THE TB AT ADDRESS CONTAINED IN LS 5
U 063C, 3022,15 ;5226           MOV MEM.DATA TO WR[0]           ;PLACE RESULT IN WR 0
                ;5227
```

```
U 063D, 452A,15 ;5228          BIC LS[#FF000000] TO WR[0]      ;CLEAR UPPER BYTE OF RESULT (UPPER BYTE
                ;5229                                          ; IS NOT PART OF TB AND IS UNPREDICT-
                ;5230                                          ; ABLE)
U 063E, 456E,15 ;5231          BIC LS[BIT23] TO WR[0]          ;DITTO FOR BIT 23
U 063F, 4540,15 ;5232          BIC LS[BIT0] TO WR[0]           ;DITTO FOR BIT 0.  WR 0 NOW CONTAINS
                ;5233                                          ; RESULT TO PRINT
U 0640, BE0C,15 ;5234          MOV WR[0] TO LS[T6.SUB]         ;RESULT NOW IN LS 6
U 0641, 8868,74 ;5235          JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
```

```
                ;5236   .PAGE   "  DEPOSIT UNIBUS MAP ROUTINE"
                ;5237   ;+++
                ;5238   ;
                ;5239   ; FUNCTIONAL DESCRIPTION:
                ;5240   ;
                ;5241   ;       This routine allows a write to the UNIBUS map portion of the TB on the
                ;5242   ;       memory controller.  The map area contains 512 decimal locations
                ;5243   ;       addressed from 200-3FF.  The remainder of the TB RAM is either unused
                ;5244   ;       or contains the Translation Buffer information.  The DEPOSIT TB command
                ;5245   ;       is used to write the TB portion of the TB RAMs.  The address specified
                ;5246   ;       with the DEPOSIT command will be the actual RAM address accessed.  This
                ;5247   ;       routine will do any shifting necessary on the address specified to pos-
                ;5248   ;       ition it correctly.  The address specified has been loaded by the con-
                ;5249   ;       sole in LS location 5 prior to executing this routine.  The address
                ;5250   ;       must be in HEX format.
                ;5251   ;         The data specified has been placed in LS 6 by the console prior to
                ;5252   ;       executing this routine.  Bits 07-01 will be the Valid, Prot, Modify,
                ;5253   ;       and Byte Offset bits (there are four Prot bits).  Bits 22-08 will
                ;5254   ;       contain the PA bits from PA 23 through PA 09 respectively.  Bits 31
                ;5255   ;       through 23 and bit 0 are unused. This routine will do any shifting nec-
                ;5256   ;       essary to write the bits in TB as described.  The data specified must
                ;5257   ;       be in HEX format.
                ;5258   ;
                ;5259   ; CALLING SEQUENCE:
                ;5260   ;
                ;5261   ;       The console loads the address of a pointer to this routine (a JMP in-
                ;5262   ;       struction at WCS location 58) into the UPC and starts the CPU.
                ;5263   ;
                ;5264   ; INPUT PARAMETERS:
                ;5265   ;
                ;5266   ;       LS 5 - UNIBUS Map address in TB (must be in range 200-3FF HEX).
                ;5267   ;       LS 6 - Contains the data to write into the UNIBUS Map in bits 1-22.
                ;5268   ;
                ;5269   ; IMPLICIT INPUTS:
                ;5270   ;
                ;5271   ;       None
                ;5272   ;
                ;5273   ; OUTPUT PARAMATERS:
                ;5274   ;
                ;5275   ;       The UNIBUS map portion of the TB RAMs gets written with the data from
                ;5276   ;       LS 6 at the address specified in LS 5.
                ;5277   ;
                ;5278   ; IMPLICIT OUTPUTS:
                ;5279   ;
                ;5280   ;       None
                ;5281   ;
                ;5282   ; SIDE EFFECTS:
                ;5283   ;
                ;5284   ;       WR 0 will be modified
                ;5285   ;       WR 1 will be modified
                ;5286   ;
                ;5287   ;---
                ;5288
                ;5289   DEPOSIT.UBS:
U 0642, 0864,DC ;5290           JSR [SHIFT.UB.ADR]              ;GO TO SUBROUTINE TO ARRANGE UNIBUS
```

```
                    ;5291                                   ; MAP ADDRESS CORRECTLY
U 0643, 360C,15     ;5292           MOV LS[T6.SUB] TO WR[0]  ;GET DATA FROM LS 6
U 0644, 4526,15     ;5293           BIC LS[#FF] TO WR[0]     ;CLEAR LOW BYTE OF DATA SPECIFIED.
                    ;5294                                   ; LOW BYTE WILL BE PUT IN BITS 24-31
                    ;5295                                   ; LATER
U 0645, 8862,AC     ;5296           JSR [SHIFT.LOOP]         ;GO TO SUBROUTINE TO SHIFT WR 0 EIGHT
                    ;5297                                   ; PLACES RIGHT
U 0646, E40C,15     ;5298           SWAP LS[T6.SUB] WITH WR[0] ;SWAP ORIGINAL DATA SPECIFIED WITH MOD-
                    ;5299                                   ; IFIED DATA.  NOW LS 6 CONTAINS BITS
                    ;5300                                   ; 8-22 IN BITS 0-14 AS REQUIRED.  WR 0
                    ;5301                                   ; CONTAINS ORIGINAL DATA SPECIFIED
U 0647, 452C,15     ;5302           BIC LS[#FFFFFF00] TO WR[0] ;CLEAR ALL BUT LOW BYTE OF ORIGINAL
                    ;5303                                   ; DATA SPECIFIED
U 0648, 8862,AC     ;5304           JSR [SHIFT.LOOP]         ;GO TO SUBROUTINE TO SHIFT WR 0 EIGHT
                    ;5305                                   ; PLACES RIGHT.  ON RETURN, WR 0 WILL
                    ;5306                                   ; HAVE BITS 0-7 IN BITS 24-31, OTHER
                    ;5307                                   ; BITS ARE CLEAR
U 0649, EDOC,15     ;5308           BIS WR[0] TO LS[T6.SUB]  ;NOW LS 6 CONTAINS DATA TO WRITE IN
                    ;5309                                   ; TB IN CORRECT FORMAT  I.E. PA BITS
                    ;5310                                   ; IN BITS 00-14 AND BYTE OFFSET, MODIFY
                    ;5311                                   ; PROT A THROUGH PROT D, AND VALID IN
                    ;5312                                   ; BITS 25-31.
                    ;5313
U 064A, 1B0B,F5     ;5314           MEM.REQ[WRITE.UBS.MAP] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO
                    ;5315                                   ; WRITE THE UNIBUS MAP
U 064B, B20C,15     ;5316           WRITE.MEM LS[T6.SUB]     ;WRITE DATA FROM LS 6 IN UNIBUS MAP
U 064C, 8868,74     ;5317           JMP [WAIT.SUB]           ;JUMP TO SET ATTN AND WAIT
                    ;5318   ;
                    ;5319   ;       SUBROUTINE TO POSITION ADDRESS SPECIFIED CORRECTLY TO ADDRESS THE
                    ;5320   ;       UNIBUS MAP.
                    ;5321   ;
                    ;5322   SHIFT.UB.ADR:
U 064D, 360A,15     ;5323           MOV LS[T5.SUB] TO WR[0]  ;GET ADDRESS SPECIFIED IN WR 0
U 064E, A2D0,15     ;5324           SHL2 WR[0]               ;SHIFT ADDRESS 2 PLACES LEFT
U 064F, A2D0,15     ;5325           SHL2 WR[0]               ;SHIFT ADDRESS 2 PLACES LEFT
U 0650, A2D0,15     ;5326           SHL2 WR[0]               ;SHIFT ADDRESS 2 PLACES LEFT
U 0651, A2D0,15     ;5327           SHL2 WR[0]               ;SHIFT ADDRESS 2 PLACES LEFT
U 0652, 23D0,15     ;5328           ASHL WR[0]               ;SHIFT LEFT ONE MORE PLACE
                    ;5329                                   ; NOW ADDRESS IS IN BITS 9-17 OF WR 0
                    ;5330           MOV WR[0] TO LS[T5.SUB], ;NOW LS 5 CONTAINS CORRECT ADDRESS.
U 0653, 3E0A,14     ;5331            RETURN                  ; RETURN TO MAIN
                    ;5332
                    ;5333
```

```
                ;5334   .PAGE   "  EXAMINE UNIBUS MAP ROUTINE"
                ;5335   ;+++
                ;5336   ;       This routine will read the UNIBUS map portion of the memory controller.
                ;5337   ;       The UNIBUS MAP contents will be returned with the byte offset, modify,
                ;5338   ;       prot A through prot D, and valid in bits 01-07 respectively.  PA bits
                ;5339   ;       09-23 are returned in bits 08-22 respectively.  Bits 23-31 and bit 0
                ;5340   ;       are not used and so will be cleared before printing the result.  The
                ;5341   ;       result will be put in LS 6 for the console to print.
                ;5342   ;         The routine will issue a memory request with memory function=READ.UBS
                ;5343   ;       .MAP and data type=LONGWORD.  The console will load LS 5 with the ad-
                ;5344   ;       dress specified before executing this routine.  The routine will shift
                ;5345   ;       the address around as required to address the UNIBUS map location spec-
                ;5346   ;       ified.  The address specified must be in HEX format.
                ;5347   ;         The UNIBUS map consists of 512 decimal locations (addresses from 200-
                ;5348   ;       3FF)  The remaining locations in the TB RAM are either unused or used
                ;5349   ;       by the translation buffer.  TB contents are read via the EXAMINE TB
                ;5350   ;       routine.
                ;5351   ;
                ;5352   ; CALLING SEQUENCE:
                ;5353   ;
                ;5354   ;       The console loads the address of a pointer to this routine (a JMP in-
                ;5355   ;       struction at WCS location 59) into the UPC and starts the CPU.
                ;5356   ;
                ;5357   ; INPUT PARAMETERS:
                ;5358   ;
                ;5359   ;       LS 5 - TB address in hex format
                ;5360   ;
                ;5361   ; IMPLICIT INPUTS:
                ;5362   ;
                ;5363   ;       None
                ;5364   ;
                ;5365   ; OUTPUT PARAMATERS:
                ;5366   ;
                ;5367   ;       LS 6 - UNIBUS map data from address specified
                ;5368   ;
                ;5369   ; IMPLICIT OUTPUTS:
                ;5370   ;
                ;5371   ;       None
                ;5372   ;
                ;5373   ; SIDE EFFECTS:
                ;5374   ;
                ;5375   ;       LS 5 is modified
                ;5376   ;       WR 0 will be modified
                ;5377   ;
                ;5378   ;---
                ;5379
                ;5380   EXAMINE.UBS:
U 0654, 0864,DC ;5381           JSR [SHIFT.UB.ADR]                  ;SHIFT ADDRESS SPECIFIED FROM LS 5 INTO
                ;5382                                               ; CORRECT POSITION AND REPLACE IN LS 5
U 0655, 1C0B,75 ;5383           MEM.REQ[READ.UBS.MAP] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO
                ;5384                                               ; READ THE UBS MAP AT ADDRESS CONTAINED
                ;5385                                               ; IN LS 5
U 0656, 3022,15 ;5386           MOV MEM.DATA TO WR[0]               ;PLACE RESULT IN WR 0
                ;5387
U 0657, 452A,15 ;5388           BIC LS[#FF000000] TO WR[0]          ;CLEAR UPPER BYTE OF RESULT (UPPER BYTE
```

```
                ;5389                                   ; IS NOT PART OF TB AND IS UNPREDICT-
                ;5390                                   ; ABLE)
U 0658, 456E,15 ;5391       BIC LS[BIT23] TO WR[0]      ;DITTO FOR BIT 23
U 0659, 4540,15 ;5392       BIC LS[BIT0] TO WR[0]       ;DITTO FOR BIT 0.  WR 0 NOW CONTAINS
                ;5393                                   ; RESULT TO PRINT
U 065A, BE0C,15 ;5394       MOV WR[0] TO LS[T6.SUB]     ;RESULT NOW IN LS 6
U 065B, 8868,74 ;5395       JMP [WAIT.SUB]              ;JUMP TO SET ATTN AND WAIT
                ;5396                                   ; CONTROL
```

```
                    ;5397   .PAGE   '' DEPOSIT MAIN MEMORY ROUTINE''
                    ;5398   ;+++
                    ;5399   ;
                    ;5400   ; FUNCTIONAL DESCRIPTION:
                    ;5401   ;
                    ;5402   ;       This routine write a longword of data into main memory at the Physical
                    ;5403   ;       address specified.  This address need not be on a longword boundary.
                    ;5404   ;       The console will use this routine for data transfers to main memory as
                    ;5405   ;       well as for the DEPOSIT MM command.  The console will have loaded the
                    ;5406   ;       address into LS 5 and the data into LS 6 before executing this routine.
                    ;5407   ;         The routine issues a memory request with the memory function=WRITE.P
                    ;5408   ;       and the data type=longword.  It then issues a WRITE.MEM LS[6] instruc-
                    ;5409   ;       tion to write the data into memory.  A check on ERROR SUM is made to
                    ;5410   ;       assure that the write occured without an error.  If ERROR SUM is as-
                    ;5411   ;       serted, the CPU will issue a CPU ACK before asserting CPU ATTN to in-
                    ;5412   ;       form the console that an error occurred.
                    ;5413   ;
                    ;5414   ; CALLING SEQUENCE:
                    ;5415   ;
                    ;5416   ;       The console loads the address of a pointer to this routine (a JMP in-
                    ;5417   ;       struction at WCS location 54) into the UPC and starts the CPU.
                    ;5418   ;
                    ;5419   ; INPUT PARAMETERS:
                    ;5420   ;
                    ;5421   ;       LS 5 - Main memory physical address
                    ;5422   ;       LS 6 - Main memory longword data
                    ;5423   ;
                    ;5424   ; IMPLICIT INPUTS:
                    ;5425   ;
                    ;5426   ;       None
                    ;5427   ;
                    ;5428   ; OUTPUT PARAMATERS:
                    ;5429   ;
                    ;5430   ;       Main Memory address specified is written with data from LS 6.
                    ;5431   ;
                    ;5432   ; IMPLICIT OUTPUTS:
                    ;5433   ;
                    ;5434   ;       A memory error asserts CPU ACK to inform the console of a write error.
                    ;5435   ;
                    ;5436   ; SIDE EFFECTS:
                    ;5437   ;
                    ;5438   ;       None
                    ;5439   ;
                    ;5440   ;---
                    ;5441
                    ;5442   DEPOSIT.MM:
U 065C, 990A,75     ;5443           MEM.REQ[WRITE.P] ADRS[T5.SUB] DT[LONG]  ;SET UP FOR WRITE TO MAIN
                    ;5444                                                   ; MEMORY USING THE PHYSICAL ADDRESS
                    ;5445                                                   ; STORED IN LS 5
                    ;5446           WRITE.MEM LS[T6.SUB],                   ;WRITE THE DATA FROM LS 6 INTO MAIN
                    ;5447                                                   ; MEMORY
U 065D, 320C,1D     ;5448            SKIP.IF[MEM.REF.OK]                    ;SKIP NEXT INSTRUCTION IF NO MEMORY
                    ;5449                                                   ; ERROR (NO ERR SUM)
U 065E, 10D0,15     ;5450           MISC [SET.CP.ACK]                       ;SET CPU ACK IF ERROR SUM ASSERTED
                    ;5451
```

```
U 065F, 3644,15 ;5452          MOV LS[#4] TO WR[0]          ;PUT A 4 IN WR 0
U 0660, 6C0A,15 ;5453          ADD WR[0] TO LS[T5.SUB]      ;INCREMENT LS 5 FOR NEXT LONGWORD DEP
U 0661, 8868,74 ;5454          JMP [WAIT.SUB]               ;JUMP TO SET ATTN AND WAIT
```

```
                ;5455   .PAGE   "  EXAMINE MAIN MEMORY ROUTINE"
                ;5456   ;+++
                ;5457   ;
                ;5458   ; FUNCTIONAL DESCRIPTION:
                ;5459   ;
                ;5460   ;       This routine is used to read a longword form main memory at the Phy-
                ;5461   ;       sical address specified.  The address need not be on a longword bound-
                ;5462   ;       ary.  This routine is used by the console on an EXAMINE.MM command.
                ;5463   ;       The console will have loaded LS 5 with the physical address specified
                ;5464   ;       before executing this routine.
                ;5465   ;         The routine will first write CSR 1 to set the INH REP CRD (inhibit
                ;5466   ;       report correctable read data) bit to prevent an error sum from occuring
                ;5467   ;       on a single bit error which has been corrected.  Then the routine is-
                ;5468   ;       sues a memory request with memory function=READ.P and DT=longword.  It
                ;5469   ;       then reads the memory location and puts the result in LS 6.  It also
                ;5470   ;       checks for an ERROR SUM and issues a CPU ACK if an error occurred (sin-
                ;5471   ;       gle bit errors that have been corrected will not cause an ERR SUM).  If
                ;5472   ;       an error occurred, CPU ACK will be set before issueing CPU ATTN to in-
                ;5473   ;       form the console that an error occurred.
                ;5474   ;
                ;5475   ; CALLING SEQUENCE:
                ;5476   ;
                ;5477   ;       The console loads the address of a pointer to this routine (a JMP in-
                ;5478   ;       struction at WCS location 55) into the UPC and starts the CPU.
                ;5479   ;
                ;5480   ; INPUT PARAMETERS:
                ;5481   ;
                ;5482   ;       LS 5 - Physical address in main memory to read.
                ;5483   ;
                ;5484   ; IMPLICIT INPUTS:
                ;5485   ;
                ;5486   ;       LS[INH.CRD] - Data to write into CSR 1 to inhibit error sum on a CRD.
                ;5487   ;       LS[CSR1]  - Address for writing CSR 1.
                ;5488   ;
                ;5489   ; OUTPUT PARAMATERS:
                ;5490   ;
                ;5491   ;       LS 6 - Longword data from physical memory address specified.
                ;5492   ;
                ;5493   ; IMPLICIT OUTPUTS:
                ;5494   ;
                ;5495   ;       CPU ACK if an error sum occurs.
                ;5496   ;
                ;5497   ; SIDE EFFECTS:
                ;5498   ;
                ;5499   ;       None
                ;5500   ;
                ;5501   ;---
                ;5502
                ;5503   EXAMINE.MM:
U 0662, 1D45,F5 ;5504           MEM.REQ[WRITE.CSR] ADRS[CSR1] DT[LONG]  ;ISSUE MEMORY REQUEST TO
                ;5505                                                   ; WRITE CSR 1
U 0663, B278,15 ;5506           WRITE.MEM LS[INH.CRD]                   ;SET INH REP CRD IN CSR 1 AND CLEAR
                ;5507                                                   ; ECC DISABLE
                ;5508
U 0664, 180B,F5 ;5509           MEM.REQ[READ.P] ADRS[T5.SUB] DT[LONG]  ;ISSUE MEMORY REQUEST TO
```

```
                ;5510                                   ; READ LONGWORD DATA FROM MAIN MEMORY
                ;5511                                   ; PHYSICAL ADDRESS IN LS 5
                ;5512       MOV MEM.DATA TO LS[T6.SUB], ;READ DATA AND PUT IN LS 6
U 0665, 380C,1D ;5513        SKIP.IF[MEM.REF.OK]        ; SKIP NEXT INSTRUCTION IF NO MEMORY
                ;5514                                   ; ERROR (NO ERROR SUM)
U 0666, 10D0,15 ;5515       MISC [SET.CP.ACK]          ;SET CPU ACK IF AN ERROR OCCURRED
U 0667, 8868,74 ;5516       JMP [WAIT.SUB]             ;JUMP TO SET ATTN AND WAIT
```

```
                    ;5517   .page   '' EXAMINE IDC ROUTINES''
                    ;5518   ;+++
                    ;5519   ;
                    ;5520   ; FUNCTIONAL DESCRIPTION:
                    ;5521   ;
                    ;5522   ;       The following routines are used to read data from the optional IDC
                    ;5523   ;       module's registers.  The result is put in LS 6 for the console to
                    ;5524   ;       print out.
                    ;5525   ;
                    ;5526   ; CALLING SEQUENCE:
                    ;5527   ;
                    ;5528   ;       The console loads the address of a pointer to this routine (a JMP in-
                    ;5529   ;       struction at WCS location 5A through 5E) into the UPC and starts the
                    ;5530   ;       CPU.
                    ;5531   ;
                    ;5532   ; INPUT PARAMETERS:
                    ;5533   ;
                    ;5534   ;       None
                    ;5535   ;
                    ;5536   ; IMPLICIT INPUTS:
                    ;5537   ;
                    ;5538   ;       None
                    ;5539   ;
                    ;5540   ; OUTPUT PARAMATERS:
                    ;5541   ;
                    ;5542   ;       LS 6 - Data from IDC register specified.
                    ;5543   ;
                    ;5544   ; IMPLICIT OUTPUTS:
                    ;5545   ;
                    ;5546   ;       None
                    ;5547   ;
                    ;5548   ; SIDE EFFECTS:
                    ;5549   ;
                    ;5550   ;       None
                    ;5551   ;
                    ;5552   ;---
                    ;5553
                    ;5554   EXAMINE.ICSR:
U 0668, 9090,15     ;5555           MISC [SET.PORT.I.0]             ;SELECT PORT TO BUS
U 0669, 9780,15     ;5556           PORT.READ PORT.ADRS[CSR]        ;SEND THE READ COMMAND
U 066A, 0867,64     ;5557           JMP [READ.IDC]                  ;READ THE DATA
                    ;5558
                    ;5559   EXAMINE.IDAR:
U 066B, 9090,15     ;5560           MISC [SET.PORT.I.0]             ;SELECT PORT TO BUS
U 066C, 1784,15     ;5561           PORT.READ PORT.ADRS[DAR]        ;SEND THE READ COMMAND
U 066D, 0867,64     ;5562           JMP [READ.IDC]                  ;READ THE DATA
                    ;5563
                    ;5564   EXAMINE.DBUF:
U 066E, 9090,15     ;5565           MISC [SET.PORT.I.0]             ;SELECT PORT TO BUS
U 066F, 978C,15     ;5566           PORT.READ PORT.ADRS[DATA.WORD]  ;SEND THE READ COMMAND
U 0670, 0867,64     ;5567           JMP [READ.IDC]                  ;READ THE DATA
                    ;5568
                    ;5569   EXAMINE.PATT:
U 0671, 9090,15     ;5570           MISC [SET.PORT.I.0]             ;SELECT PORT TO BUS
U 0672, 1790,15     ;5571           PORT.READ PORT.ADRS[PATTERN]    ;SEND THE READ COMMAND
```

```
U 0673, 0867,64 ;5572           JMP [READ.IDC]                  ;READ THE DATA
                ;5573
                ;5574   EXAMINE.POSIT:
U 0674, 9090,15 ;5575           MISC [SET.PORT.I.0]             ;SELECT PORT TO BUS
U 0675, 9794,15 ;5576           PORT.READ PORT.ADRS[POSITION]   ;SEND THE READ COMMAND
                ;5577
                ;5578
                ;5579   READ.IDC:
U 0676, 3A0C,15 ;5580           MOV PORT TO LS[T6.SUB]          ;READ DATA AND PUT IN LS 6
U 0677, 1080,15 ;5581           MISC [SET.ACC.I.0]              ;RETURN SELECT TO ACCELERATOR
U 0678, 8868,74 ;5582           JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
```

```
                    ;5583   .page   '' DEPOSIT IDC ROUTINES''
                    ;5584   ;+++
                    ;5585   ;
                    ;5586   ; FUNCTIONAL DESCRIPTION:
                    ;5587   ;
                    ;5588   ;       The following routines are used to write data to the optional IDC
                    ;5589   ;       module's registers.  The data is taken from LS 6 which is previously
                    ;5590   ;       loaded by the monitor when the DEPOSIT command is executed.
                    ;5591   ;
                    ;5592   ; CALLING SEQUENCE:
                    ;5593   ;
                    ;5594   ;       The console loads the address of a pointer to this routine (a JMP in-
                    ;5595   ;       struction at WCS location 5F through 61) into the UPC and starts the
                    ;5596   ;       CPU.
                    ;5597   ;
                    ;5598   ; INPUT PARAMETERS:
                    ;5599   ;
                    ;5600   ;       LS 6 - Data pattern to write.
                    ;5601   ;
                    ;5602   ; IMPLICIT INPUTS:
                    ;5603   ;
                    ;5604   ;       None
                    ;5605   ;
                    ;5606   ; OUTPUT PARAMATERS:
                    ;5607   ;
                    ;5608   ;       None
                    ;5609   ;
                    ;5610   ; IMPLICIT OUTPUTS:
                    ;5611   ;
                    ;5612   ;       None
                    ;5613   ;
                    ;5614   ; SIDE EFFECTS:
                    ;5615   ;
                    ;5616   ;       None
                    ;5617   ;
                    ;5618   ;---
                    ;5619
                    ;5620   DEPOSIT.ICSR:
U 0679, 360C,15     ;5621           MOV LS[T6.SUB] TO WR[0]          ;GET DATA PATTERN TO WRITE
U 067A, 17A0,15     ;5622           PORT.WRITE PORT.ADRS[CSR] WITH WR[0]    ;WRITE TO IDC
U 067B, 8868,74     ;5623           JMP [WAIT.SUB]                   ;JUMP TO SET ATTN AND WAIT
                    ;5624
                    ;5625
                    ;5626   DEPOSIT.IDAR:
U 067C, 360C,15     ;5627           MOV LS[T6.SUB] TO WR[0]          ;GET DATA PATTERN TO WRITE
U 067D, 97A4,15     ;5628           PORT.WRITE PORT.ADRS[DAR] WITH WR[0]    ;WRITE TO IDC
U 067E, 8868,74     ;5629           JMP [WAIT.SUB]                   ;JUMP TO SET ATTN AND WAIT
                    ;5630
                    ;5631
                    ;5632   DEPOSIT.DBUF:
U 067F, 360C,15     ;5633           MOV LS[T6.SUB] TO WR[0]          ;GET DATA PATTERN TO WRITE
U 0680, 17AC,15     ;5634           PORT.WRITE PORT.ADRS[DATA.WORD] WITH WR[0]      ;WRITE TO IDC
U 0681, 8868,74     ;5635           JMP [WAIT.SUB]                   ;JUMP TO SET ATTN AND WAIT
                    ;5636
                    ;5637
```

```
                    ;5638   CLEAR.FIFO.ADDR:
U 0682, 17C0,15     ;5639           PORT.CONTROL [CLEAR.FIFO.CNTR]  ;CLEAR ADDRESS IN FIFO A OR FIFO B
U 0683, 8868,74     ;5640           JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
                    ;5641
                    ;5642
                    ;5643   SELECT.FIFO.A:
U 0684, 17D8,15     ;5644           PORT.CONTROL [SEL.FIFO.A]       ;SELECT FIFO A
U 0685, 8868,74     ;5645           JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
                    ;5646
                    ;5647
                    ;5648   SELECT.FIFO.B:
U 0686, 97DC,15     ;5649           PORT.CONTROL [SEL.FIFO.B]       ;SELECT FIFO B
                    ;5650
                    ;5651   WAIT.SUB:
U 0687, 10E0,15     ;5652           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;5653   WAIT.DE.EX:
U 0688, 8868,84     ;5654           JMP [WAIT.DE.EX]                ;LOOP SELF UNTIL CONSOLE TAKES CONTROL
```

```
                ;5655   .PAGE   "  SAVE WR ROUTINE"
                ;5656   ;+++
                ;5657   ;
                ;5658   ; FUNCTIONAL DESCRIPTION:
                ;5659   ;
                ;5660   ;       This routine saves WRs 0-3 in LS locations 1-4.  When the console takes
                ;5661   ;       control of the CPU, it calls this routine so that the working registers
                ;5662   ;       can be used by the console.  The WRs are restored by another routine
                ;5663   ;       called by the console before the CPU resumes execution (on a COntinue
                ;5664   ;       command, for example).
                ;5665   ;         This routine simply moves data from WR 0-3 to LS locations 1-4 and
                ;5666   ;       then issues a CPU ATTN to the console.  It then loops on a JMP self
                ;5667   ;       until the console takes over.
                ;5668   ;
                ;5669   ; CALLING SEQUENCE:
                ;5670   ;
                ;5671   ;       The console loads the address of a pointer to this routine (a JMP in-
                ;5672   ;       struction at WCS location 46) into the UPC and starts the CPU.
                ;5673   ;
                ;5674   ; INPUT PARAMETERS:
                ;5675   ;
                ;5676   ;       None
                ;5677   ;
                ;5678   ; IMPLICIT INPUTS:
                ;5679   ;
                ;5680   ;       None
                ;5681   ;
                ;5682   ; OUTPUT PARAMATERS:
                ;5683   ;
                ;5684   ;       LS 1 - Contents of WR 0
                ;5685   ;       LS 2 - Contents of WR 1
                ;5686   ;       LS 3 - Contents of WR 2
                ;5687   ;       LS 4 - Contents of WR 3
                ;5688   ;
                ;5689   ; IMPLICIT OUTPUTS:
                ;5690   ;
                ;5691   ;       None
                ;5692   ;
                ;5693   ; SIDE EFFECTS:
                ;5694   ;
                ;5695   ;       None
                ;5696   ;
                ;5697   ;---
                ;5698
                ;5699   SAVE.WR:
                ;5700
U 0689, 3E02,15 ;5701           MOV WR[0] TO LS[SAV.WR0]        ;SAVE WR 0 IN LS LOCATION 1
U 068A, BE04,95 ;5702           MOV WR[1] TO LS[SAV.WR1]        ;SAVE WR 1 IN LS LOCATION 2
U 068B, 3E07,15 ;5703           MOV WR[2] TO LS[SAV.WR2]        ;SAVE WR 2 IN LS LOCATION 3
U 068C, 3E09,95 ;5704           MOV WR[3] TO LS[SAV.WR3]        ;SAVE WR 3 IN LS LOCATION 4
U 068D, 8868,74 ;5705           JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
```

```
                ;5706   .PAGE   '' RESTORE WR ROUTINE''
                ;5707   ;+++
                ;5708   ;
                ;5709   ; FUNCTIONAL DESCRIPTION:
                ;5710   ;
                ;5711   ;       This routine restores the 2901 working registers save in LS 1-4.  The
                ;5712   ;       console calls this routine before the CPU is restarted after it has
                ;5713   ;       been halted by the console.
                ;5714   ;         The routine simply moves the data from LS locations 1-4 to WRs 0-3
                ;5715   ;       and then issues a CPU ATTN to the console.  A JMP self is executed next
                ;5716   ;       waiting for the console to take control.
                ;5717   ;
                ;5718   ; CALLING SEQUENCE:
                ;5719   ;
                ;5720   ;       The console loads the address of a pointer to this routine (a JMP in-
                ;5721   ;       struction at WCS location 47) into the UPC and starts the CPU.
                ;5722   ;
                ;5723   ; INPUT PARAMETERS:
                ;5724   ;
                ;5725   ;       LS 1 contains the data to be restored in WR 0.
                ;5726   ;       LS 2 contains the data to be restored in WR 1.
                ;5727   ;       LS 3 contains the data to be restored in WR 2.
                ;5728   ;       LS 4 contains the data to be restored in WR 3.
                ;5729   ;
                ;5730   ; IMPLICIT INPUTS:
                ;5731   ;
                ;5732   ;       None
                ;5733   ;
                ;5734   ; OUTPUT PARAMATERS:
                ;5735   ;
                ;5736   ;       WR 0 gets data from LS 1.
                ;5737   ;       WR 1 gets data from LS 2.
                ;5738   ;       WR 2 gets data from LS 3.
                ;5739   ;       WR 3 gets data from LS 4.
                ;5740   ;
                ;5741   ; IMPLICIT OUTPUTS:
                ;5742   ;
                ;5743   ;       None
                ;5744   ;
                ;5745   ; SIDE EFFECTS:
                ;5746   ;
                ;5747   ;       None
                ;5748   ;
                ;5749   ;---
                ;5750
                ;5751   RESTORE.WR:
U 068E, B602,15 ;5752           MOV LS[SAV.WR0] TO WR[0]        ;RESTORE WR 0
U 068F, 3604,95 ;5753           MOV LS[SAV.WR1] TO WR[1]        ;RESTORE WR 1
U 0690, B607,15 ;5754           MOV LS[SAV.WR2] TO WR[2]        ;RESTORE WR 2
U 0691, B609,95 ;5755           MOV LS[SAV.WR3] TO WR[3]        ;RESTORE WR 3
U 0692, 8868,74 ;5756           JMP [WAIT.SUB]                  ;JUMP TO SET ATTN AND WAIT
```

```
;5757   .PAGE   "WCS SUBROUTINES FOR CPU USE"
;5758   .TOC    "  ERROR ROUTINE"
;5759   ;+++
;5760   ;
;5761   ; FUNCTIONAL DESCRIPTION:
;5762   ;
;5763   ;       This routine is used to detect and report diagnostic errors occurring
;5764   ;       during WCS based micro-diagnostics.  The WCS micro-code sets up the
;5765   ;       parameters listed below and calls this routine.  The routine compares
;5766   ;       WR 1 (expected data) and WR 0 (received data) according to the error
;5767   ;       mask.  Bits set in the mask indicate bits that are not checked for
;5768   ;       errors.
;5769   ;        If WR 0 and WR 1 are equal (no error) the routine executes a return+1
;5770   ;       to the calling point.  The only exception to this is if loop on error
;5771   ;       is set.  In that case, the error number is checked to see if that error
;5772   ;       occurred previously.  If so, then the error loop is forced by doing a
;5773   ;       return.  This will lock an error loop in on intermittent errors.
;5774   ;        If WR 0 and WR 1 are not equal (an error) then bit 8 is set in the
;5775   ;       CONTROL AND STATUS word, expected and received data is set up in LS for
;5776   ;       printout, and flags are checked in the ERROR CONTROL word.  This con-
;5777   ;       trols whether printouts are disabled, loop on error is enabled, etc.
;5778   ;       After printing the error, a return+1 is made to the calling point.
;5779   ;        Loop on error causes a return rather than return+1 to cause an error
;5780   ;       loop.  Also CPU ACKNOWLEDGE is toggled at the end of the routine for
;5781   ;       a scope sync point.
;5782   ;
;5783   ; CALLING SEQUENCE:
;5784   ;
;5785   ;       JSR [CHECK.RESULT]
;5786   ;
;5787   ; INPUT PARAMETERS:
;5788   ;
;5789   ;       WR[0] = Received data i.e. the actual result of the test
;5790   ;       WR[1] = Expected data i.e. what the result should have been
;5791   ;
;5792   ; IMPLICIT INPUTS:
;5793   ;
;5794   ;       LS[ERROR.MASK] = Used to mask out bits that are not to be tested
;5795   ;       LS[ERROR.NUMBER] = Current error number
;5796   ;       LS[PREVIOUS.ERROR] = Error number of previous data
;5797   ;       LS[CONTROL.STATUS] = Control and Status word.  Contains information
;5798   ;           written by the CPU to inform the monitor what to do.
;5799   ;       LS[ERROR.CONTROL] = Information such as loop-on-error, bell-on-error,
;5800   ;           no error report, and halt on error.
;5801   ;
;5802   ; OUTPUT PARAMATERS:
;5803   ;
;5804   ;       NONE
;5805   ;
;5806   ; IMPLICIT OUTPUTS:
;5807   ;
;5808   ;       LS[CONTROL.STATUS] = Control and Status with new status information
;5809   ;       LS[PREVIOUS.ERROR] = Last error number (modified only if error and
;5810   ;           loop on error is enabled)
;5811   ;       LS[EXPECTED.DATA] = Expected result if an error was detected
```

```
                    ;5812   ;       LS[RECEIVED.DATA] = Actual result if an error was detected
                    ;5813   ;
                    ;5814   ; SIDE EFFECTS:
                    ;5815   ;
                    ;5816   ;       WR[0], WR[1], and the Q reg are modified and are not restored before
                    ;5817   ;           returning.
                    ;5818   ;       If loop-on-error is enabled and an error loop is to be executed, CPU
                    ;5819   ;           ACKNOWLEDGE will be toggled for a scope sync.
                    ;5820   ;
                    ;5821   ;---
                    ;5822
                    ;5823   CHECK.RESULT:
U 0693, 458A,15     ;5824           BIC LS[ERROR.MASK] TO WR[0]     ;MASK OFF NOT TESTED BITS (CLEAR THEM)
                    ;5825                                           ; FOR RECEIVED DATA
U 0694, C58A,95     ;5826           BIC LS[ERROR.MASK] TO WR[1]     ;DITTO FOR EXPECTED DATA
                    ;5827
                    ;5828           XOR WR[0] WITH WR[1] TO Q,      ;CMP RECEIVED DATA IN WR[0] WITH
U 0695, AB80,B5     ;5829            DT(LONG)&SET.ALU.CC            ; EXPECTED DATA IN WR[1] FOR ERROR
U 0696, 8869,F1     ;5830           JMP.IF[NEQ] TO [ERROR]          ;JUMP IF ERROR
                    ;5831
U 0697, 3690,15     ;5832           MOV LS[ERROR.CONTROL] TO WR[0]  ;GET ERROR CONTROL WORD FROM LS
                    ;5833           BIT WR[0] WITH LS[LOE],         ;SEE IF LOOP ON ERROR IS ENABLED
U 0698, 5940,35     ;5834            DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 0699, DB00,01     ;5835           SKIP.IF[BIT.SET]                ;SKIP IF IT IS
U 069A, DB00,16     ;5836           RETURN+1                        ;ELSE RETURN+1 (NO ERROR AND NO LOOP)
                    ;5837
U 069B, 36A0,15     ;5838           MOV LS[PREVIOUS.ERROR] TO WR[0] ;IF NO ERROR BUT LOOP ON ERROR IS
                    ;5839                                           ; ENABLED, SEE IF THERE WAS A
                    ;5840                                           ; PREVIOUS ERROR
                    ;5841           XOR WR[0] WITH LS[ERROR.NUMBER] TO Q,
U 069C, CD82,35     ;5842            DT(LONG)&SET.ALU.CC            ;IS THIS THE SAME SPOT THAT HAD AN
                    ;5843                                           ; ERROR BEFORE? (INTERMITTANT ERROR)
U 069D, 086B,91     ;5844           JMP.IF[NEQ] TO [RET+1]          ;JUMP IF NOT TO RETURN WITHOUT ERROR
                    ;5845                                           ; LOOPING (RETURN+1)
U 069E, 086B,C4     ;5846           JMP [RET]                       ;ELSE CONTINUE TO LOOP (WE WILL LOOP
                    ;5847                                           ; ON ERROR EVEN IF IT GOES AWAY)
                    ;5848
                    ;5849   ERROR:
U 069F, 3E86,15     ;5850           MOV WR[0] TO LS[RECEIVED.DATA]  ;PUT RESULT OF TEST IN LS FOR PRINTOUT
U 06A0, 3690,15     ;5851           MOV LS[ERROR.CONTROL] TO WR[0]  ;GET ERROR CONROL BITS TO CHECK FOR
                    ;5852                                           ; LOOP COMMAND
                    ;5853           BIT WR[0] WITH LS[LOOP.COMMAND],;SEE IF BIT 6 SET
U 06A1, 594C,35     ;5854            DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 06A2, 086B,C1     ;5855           JMP.IF[BIT.SET] TO [RET]        ;GO LOOP IF SET (FAST LOOP)
                    ;5856
U 06A3, 3E84,95     ;5857           MOV WR[1] TO LS[EXPECTED.DATA]  ;PUT EXPECTED DATA IN LS FOR PRINTOUT
                    ;5858
U 06A4, 3680,95     ;5859           MOV LS[CONTROL.STATUS] TO WR[1] ;GET CONTROL AND STATUS WORD FROM LS
U 06A5, C532,95     ;5860           BIC LS[#FE7FFFFF] TO WR[1]      ;CLEAR ALL BUT BITS 23&24 IN CONTROL
                    ;5861                                           ; AND STATUS WORD
U 06A6, C750,95     ;5862           BIS LS[ERROR] TO WR[1]          ;SET BIT 8 IN CONTROL AND STATUS WORD
                    ;5863                                           ; (INDICATES AN ERROR WAS DETECTED)
U 06A7, BE80,95     ;5864           MOV WR[1] TO LS[CONTROL.STATUS] ;WRITE UPDATED CONTROL AND STATUS WORD
                    ;5865                                           ; BACK TO LS[40]
                    ;5866
```

```
                ;5867           BIT WR[0] WITH LS[BELL],        ;SEE IF BELL ON ERROR IS SET (WR 0 STILL
                ;5868                                           ; CONTAINS ERROR CONTROL WORD)
U 06A8, D944,35 ;5869            DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 06A9, 886A,D9 ;5870           JMP.IF[BITS.CLR] TO [CHECK.NER] ;IF BELL ON ERROR IS NOT SET, JUMP TO
                ;5871                                           ; SEE IF ERROR REPORTING IS INHIBITED
U 06AA, 10E0,15 ;5872           MISC [SET.CP.ATTN]              ;ELSE SET CPU ATTN (RING MY BELL)
                ;5873
U 06AB, 086A,B4 ;5874   WAIT.1: JMP [WAIT.1]                    ;WAIT FOR CONSOLE TO STOP ME
U 06AC, 086B,64 ;5875           JMP [LOOP]                      ;GO TO CHECK LOOP-ON-ERROR AFTER
                ;5876                                           ; CONSOLE HAS FINISHED
                ;5877
                ;5878   CHECK.NER:
                ;5879           BIT WR[0] WITH LS[NER],         ;IF NO BELL SEE IF ERROR REPORT IS
                ;5880                                           ; INHIBITED
U 06AD, D942,35 ;5881            DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 06AE, 886B,21 ;5882           JMP.IF[BIT.SET] TO [CHECK.HALT] ;JUMP IF ERROR REPORT IS INHIBITED TO
                ;5883                                           ; CHECK FOR HALT ON ERROR
                ;5884
U 06AF, 10E0,15 ;5885           MISC [SET.CP.ATTN]              ;SET CPU ATTN (REPORT ERROR)
U 06B0, 086B,04 ;5886   WAIT.2: JMP [WAIT.2]                    ;WAIT FOR CONSOLE TO STOP ME
U 06B1, 086B,64 ;5887           JMP [LOOP]                      ;GO TO CHECK LOOP-ON-ERROR AFTER
                ;5888                                           ; CONSOLE HAS FINISHED
                ;5889
                ;5890   CHECK.HALT:
                ;5891           BIT WR[0] WITH LS[HOE],         ;SEE IF HALT ON ERROR IS ENABLED
U 06B2, 5946,35 ;5892            DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 06B3, 886B,69 ;5893           JMP.IF[BITS.CLR] TO [LOOP]      ;JUMP IF NOT TO CHECK LOOP-ON-ERROR
                ;5894
U 06B4, 10E0,15 ;5895           MISC [SET.CP.ATTN]              ;ELSE SET CPU ATTN (HALT ON ERROR)
U 06B5, 086B,54 ;5896   WAIT.3: JMP [WAIT.3]                    ;WAIT FOR CONSOLE TO STOP ME
                ;5897
U 06B6, 3690,15 ;5898   LOOP:   MOV LS[ERROR.CONTROL] TO WR[0]  ;GET ERROR CONTROL BITS (NER, HOE ETC.)
                ;5899           BIT WR[0] WITH LS[LOE],         ;SEE IF LOOP-ON-ERROR
U 06B7, 5940,35 ;5900            DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 06B8, DB00,01 ;5901           SKIP.IF[BIT.SET]                ; SKIP IF LOOP ON ERROR IS ENABLED
                ;5902
U 06B9, DB00,16 ;5903   RET+1:  RETURN+1                        ;ELSE RETURN+1 FOR NO ERROR LOOP
                ;5904
U 06BA, 3682,15 ;5905           MOV LS[ERROR.NUMBER] TO WR[0]   ;GET ERROR NUMBER IF LOOPING
                ;5906
U 06BB, BEA0,15 ;5907           MOV WR[0] TO LS[PREVIOUS.ERROR] ;AND SAVE IT IN PREVIOUS ERROR
                ;5908
U 06BC, 10D0,15 ;5909   RET:    MISC [SET.CP.ACK]               ;SET CPU ACKNOWLEDGE FOR SCOPE SYNC
                ;5910           MISC [CLR.CP.ATTN.AND.ACK],     ;AND THEN CLEAR IT
U 06BD, 10C0,14 ;5911            RETURN                         ; RETURN TO TEST AND LOOP ON ERROR
                ;5912
```

```
                    ;5913   .PAGE
                    ;5914   ;
                    ;5915   ;
                    ;5916   ;
                    ;5917   ERR.NO.TEST:
U 06BE, DB00,15     ;5918           NOP                                 ;NOP WILL CAUSE A SEQUENCE ERROR
U 06BF, B65E,15     ;5919           MOV LS[BEGIN.TEST] TO WR[0]         ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;5920                                               ; STATUS WORD
U 06C0, 3E80,15     ;5921           MOV WR[0] TO LS[CONTROL.STATUS]      ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;5922                                               ; BIT 15 INDICATES START OF TEST TO
                    ;5923                                               ; CONSOLE
U 06C1, 8868,74     ;5924           JMP [WAIT.SUB]                      ;JUMP TO SET ATTN AND WAIT
                    ;5925   .TOC    "START TRANSFER ROUTINE"
                    ;5926   ;+++
                    ;5927   ;       This routine loads the LS with information neccesary for these tests
                    ;5928   ;       to run.  It will then issue a CPU ATTN with the control and status word
                    ;5929   ;       clear to indicate that all transfers are complete.
                    ;5930   ;
                    ;5931   ;       RESULT: LS LOADED WITH CONSTANTS. MONITOR INFORMED TRANSFERS OVER.
                    ;5932   ;---
                    ;5933
                    ;5934   TRANSFER.POINT:
                    ;5935
                    ;5936
U 06C2, 2F80,15     ;5937           CLR WR[0]                           ;START WITH 0 IN WR[0]
U 06C3, BFFE,15     ;5938           MOV WR[0] TO LS[PSL.HW]              ;MAKE SURE COMPAT MODE IS CLEAR
                    ;5939
U 06C4, 2040,15     ;5940           INC WR[0]                           ;NOW HAVE 1 IN WR[0]
U 06C5, 23D0,15     ;5941           ASHL WR[0]                          ;10(B)
U 06C6, 2040,15     ;5942           INC WR[0]                           ;11(B)
U 06C7, 23D0,15     ;5943           ASHL WR[0]                          ;110(B)
U 06C8, 2040,15     ;5944           INC WR[0]                           ;111(B)
U 06C9, 23D0,15     ;5945           ASHL WR[0]                          ;1110(B)
U 06CA, 23D0,15     ;5946           ASHL WR[0]                          ;1 1100(B)
U 06CB, 23D0,15     ;5947           ASHL WR[0]                          ;11 1000(B)
U 06CC, 23D0,15     ;5948           ASHL WR[0]                          ;111 0000(B)  NOW HAVE 70(H) IN WR 0.
                    ;5949                                               ; THIS IS CORRECT START ADDRESS OF DATA
                    ;5950                                               ; SECTION (INCLUDING LONGWORD COUNT,
                    ;5951                                               ; DESTINATION, AND DESTINATION ADDRESS)
U 06CD, 3E0E,15     ;5952           MOV WR[0] TO LS[TRANSFER.POINTER]    ;LOAD LS 7 WITH POINTER TO TRANSFER
                    ;5953                                               ; DATA SECTION.
                    ;5954
U 06CE, 2F80,15     ;5955           CLR WR[0]                           ;0 IN WR 0
U 06CF, 2040,15     ;5956           INC WR[0]                           ;1 IN WR 0
U 06D0, 23D0,15     ;5957           ASHL WR[0]                          ;10(B)
U 06D1, 23D0,15     ;5958           ASHL WR[0]                          ;100(B)
U 06D2, 23D0,15     ;5959           ASHL WR[0]                          ;1000(B)
U 06D3, 23D0,15     ;5960           ASHL WR[0]                          ;1 0000(B)
U 06D4, 23D0,15     ;5961           ASHL WR[0]                          ;10 0000(B)
U 06D5, 23D0,15     ;5962           ASHL WR[0]                          ;100 0000(B)
U 06D6, 23D0,15     ;5963           ASHL WR[0]                          ;1000 0000(B)
U 06D7, 23D0,15     ;5964           ASHL WR[0]                          ;1 0000 0000(B)
U 06D8, 23D0,15     ;5965           ASHL WR[0]                          ;10 0000 0000(B)
U 06D9, 23D0,15     ;5966           ASHL WR[0]                          ;100 0000 0000(B)
U 06DA, 23D0,15     ;5967           ASHL WR[0]                          ;1000 0000 0000(B)
```

```
U 06DB, 23D0,15 ;5968          ASHL WR[0]                          ;1 0000 0000 0000(B)
U 06DC, 23D0,15 ;5969          ASHL WR[0]                          ;10 0000 0000 0000(B)
U 06DD, 3E80,15 ;5970          MOV WR[0] TO LS[CONTROL.STATUS]      ;SET BIT 13 IN LS CONTROL AND STATUS
                ;5971                                               ; WORD (BIT 13 INDICATES CONSOLE MUST
                ;5972                                               ; PERFORM A DATA TRANSFER)
U 06DE, 10E0,15 ;5973          MISC [SET.CP.ATTN]                  ;ISSUE CPU ATTN TO CONSOLE (DO FIRST LS
                ;5974                                               ; TRANSFER TO LOCATIONS 0 THROUGH 77(H))
                ;5975    WAIT.XFER1:
U 06DF, 086D,F4 ;5976          JMP [WAIT.XFER1]                    ;LOOP HERE WAITING FOR CONSOLE TO RE-
                ;5977                                               ; SPOND
                ;5978
U 06E0, 36CA,15 ;5979          MOV LS[#162(H)] TO WR[0]            ;GET START ADDRESS OF SECOND HALF OF LS
                ;5980                                               ; TRANSFER
U 06E1, 3E0E,15 ;5981          MOV WR[0] TO LS[TRANSFER.POINTER]    ;LOAD START ADDRESS IN LS FOR CONSOLE
U 06E2, 365A,15 ;5982          MOV LS[XFER.DATA] TO WR[0]          ;BIT 13 INDICATES TO DO DATA TRANSFER
U 06E3, 3E80,15 ;5983          MOV WR[0] TO LS[CONTROL.STATUS]      ;SET UP CONTROL AND STATUS WORD
U 06E4, 10E0,15 ;5984          MISC [SET.CP.ATTN]                  ;ISSUE CPU ATTN TO CONSOLE (DO SECOND LS
                ;5985                                               ; TRANSFER TO LOCATIONS 80 THROUGH F7(H))
                ;5986    WAIT.XFER2:
U 06E5, 086E,54 ;5987          JMP [WAIT.XFER2]                    ;LOOP HERE WAITING FOR CONSOLE TO RE-
                ;5988                                               ; SPOND
                ;5989
U 06E6, 3022,15 ;5990          MOV MEM.DATA TO WR[0]               ;THIS INSTRUCTION USED TO FREE MEMORY
                ;5991                                               ; IF HUNG WAITING FOR CPU DR IN AN
                ;5992                                               ;  OCTA FLOW
U 06E7, 3022,15 ;5993          MOV MEM.DATA TO WR[0]               ;NEEDS 4 OF THESE
U 06E8, 3022,15 ;5994          MOV MEM.DATA TO WR[0]
U 06E9, 3022,15 ;5995          MOV MEM.DATA TO WR[0]
                ;5996
U 06EA, 17CC,15 ;5997          PORT.CONTROL [CLEAR.IDC]            ;JAM IDC TO IDLE
U 06EB, 17CC,15 ;5998          PORT.CONTROL [CLEAR.IDC]
U 06EC, 17CC,15 ;5999          PORT.CONTROL [CLEAR.IDC]
U 06ED, 17CC,15 ;6000          PORT.CONTROL [CLEAR.IDC]            ;DONE JAMMING IDC
                ;6001
U 06EE, B64E,95 ;6002          MOV LS[#80] TO WR[1]                ;SET BIT 7 OF WR 1
U 06EF, 97A0,95 ;6003          PORT.WRITE PORT.ADRS[CSR] WITH WR[1] ; CLEAR CRDY IN IDC
U 06F0, 17D4,15 ;6004          PORT.CONTROL [CLEAR.AUTO]           ;CLEAR AUTO MODE IN IDC
U 06F1, 97C4,15 ;6005          PORT.CONTROL [RESET.BR]             ;CLEAR BR 7 IN IDC
                ;6006
U 06F2, 3660,15 ;6007          MOV LS[INTERUPT.EN] TO WR[0]        ;SET BIT 16 IN WR
U 06F3, 3E80,15 ;6008          MOV WR[0] TO LS[CONTROL.STATUS]      ;SET BIT 16 TO CLEAR ANY INTERRUPTS
                ;6009                                               ; AND INFORM CONSOLE THAT XFERS ARE
                ;6010                                               ; DONE
U 06F4, 10E0,15 ;6011          MISC [SET.CP.ATTN]                  ;CALL CONSOLE
                ;6012    WAIT.XFER:
U 06F5, 886F,54 ;6013          JMP [WAIT.XFER]                     ;WAIT FOR CONSOLE TO RESPOND
U 06F6, 5B00,14 ;6014          RETURN                              ;USED WHEN TRANSFER IS CALLED FROM
                ;6015                                               ; MICRO-DIAGNOSTIC ONLY
                ;6016    .LIST
```

```
;6017    .PAGE
;6018    .REGION/700,3FFF
;6019
;6020    .TOC    "MCT CSR BIT ALLOCATIONS"
;6021    ;
;6022    ;
;6023    ; CSR 0
;6024    ;
;6025    ;       CSR 0 contains 7 read only bits in positions 6-0 corresponding to
;6026    ;       the checkbits or syndrome bits from the ECC chips.  The other bits
;6027    ;       are not used and are masked when an EX MC 0 (examine MCT CSR 0) is
;6028    ;       executed.  The 7 bits are arranged as follows:
;6029    ;
;6030    ;                       BIT  6  5  4  3  2  1  0
;6031    ;                          ,--------------------,
;6032    ;       CKBT or SYND       |T |32|16|8 |4 |2 |1 |
;6033    ;                          '--------------------'
;6034    ;
;6035    ; CSR 1
;6036    ;
;6037    ;       CSR 1 contains maintenance, error, and control bits used to control
;6038    ;       or report conditions that occur in the MCT during operations.  The
;6039    ;       error bits are read only, the maintenance CKBTs are write only, and
;6040    ;       the control bits are read/write.  Two of the control bits (DIAG CK
;6041    ;       and ECC DIS) are also used as branch conditions.  These two bits are
;6042    ;       used for testing the MCT and ECC logic only.  Bits 24 and 13-07 are
;6043    ;       unused and are masked when an EX MC 1 (examine mct CSR 1) is executed.
;6044    ;       The bits are arranged as follows:
;6045    ;
;6046    ;       31     28       24       20       16       12       8   6   4       0
;6047    ;       ,-----------------------------------------------------------------------,
;6048    ;       |RDSR|CDR|TBD|IHNIE|MMIAC|DEC  |MOD|ACD|TBC|OPC|WRP|ADR|UBS|NXM|TBP|VALID|
;6049    ;       |    |   |   |     |     |     |CKD|   |   |   |   |   |   |   |   |     |
;6050    ;       |    |   |PAR|     |     |     |   |MEM|IRR|RPE|RSR|EGS|   |PAR|   |     |
;6051    ;       |    |   |ACR|     |GD   |     |RER|EIE|SRG|   |   |   |BSY|   |ERR|     |
;6052    ;       |    |   |RRD|CIS  |KS   |     |EFF|S  |   |   |   |   |   |   |ERR|     |
;6053    ;       |    |   |   |     |     |     |   |   |   |   |   |   |   |   |R  |     |
;6054    ;       |    |   |DIAG|    |     |     |   |   |   |   |   |   |   |   |   |     |
;6055    ;       |    |   |   |     |     |     |   |   |   |   |   |   |   |   |   |     |
;6056    ;       |    |   |   |     |     |     |   |   |   |   |   |   |   |   |   |     |
;6057    ;       |    |   |   |     |     |     |   |   |   |   |   |   |   |   |   |     |
```

```
;6062   .PAGE
;6063   ; CSR 2
;6064   ;
;6065   ;       CSR 2 is used to report UNIBUS errors.  Only 4 bits are used.  The
;6066   ;       other bits are masked when a EX MC 2 (examine MCT CSR 2) is executed.
;6067   ;       The bits that are used are bits 16-14 and bit 31.  The bits are arranged
;6068   ;       as follows:
;6069   ;
;607    ;               31 .... 16  14 ... 0
;6071   ;               ,-----------------.
;6072   ;               |R| ... |N|T|W| ... |
;6073   ;               |D| ... |X|B|R| ... |
;6074   ;               |S| ... |M| | | ... |
;6075   ;               | | ... | |P|N| ... |
;6076   ;               | | ... | |A|O| ... |
;6077   ;               | | ... | |R|T| ... |
;6078   ;               | | ... | | | | ... |
;6079   ;               | | ... | |E|V| ... |
;6080   ;               | | ... | |R|A| ... |
;6081   ;               | | ... | |R|L| ... |
;6082   ;               `-----------------'
;6083   ;
```

```
;6084    .page 'Memory Initialization''
;6085    ;
;6086    ;           The memory requires little initialization other than writing a
;6087    ;       data pattern throughout the main memory array to set up the checkbits
;6088    ;       for each longword.  This is done before the array tests in the micro-
;6089    ;       diagnostic at the same time that the memory is sized.
;6090    ;           A sequence of CPU instructions is executed in test 0 (which is
;6091    ;       executed automatically whenever a new section is loaded) to free up the
;6092    ;       memory controller if it is hung.  The sequence is four MOV MEM.DATA TO
;6093    ;       WR instructions followed by a READ.V.RCHK.IFILL instruction.  The READ
;6094    ;       .V.RCHK.IFILL forces a DATA RCVD in case the memory is looping waiting
;6095    ;       for CPU GRANT to go away or DATA RCVD to come up.  The four MOV MEM
;6096    ;       .DATA instructions perform this function also in case the MCT is hung
;6097    ;       in an OCTA flow.  This sequence can be forced by doing the monitor com-
;6098    ;       mand INIT if the memory should become hung during testing.
;6099    ;
;6100    .toc    'Memory Dispatch Description''
;6101    ;
;6102    ;           This section describes the sequence of events that occur when a
;6103    ;       MEM.REQ u-instruction is issued by the CPU.  While the memory is idle,
;6104    ;       it is looping on a single u-word which is clocking the VAR with what-
;6105    ;       ever is on the UNIBUS address lines. During this loop, the signal CPU
;6106    ;       GRANT L is high (not asserted).
;6107    ;           The MEM.REQ instruction generates MEMORY REQUEST H on the CPU
;6108    ;       module and sends it to the MCT module.  The CPU will stall until CPU
;6109    ;       GRANT is asserted low (actually, unless the memory is busy, grant will
;6110    ;       be asserted almost immediately and the CPU will not stall).  At the
;6111    ;       memory controller, the micro state after the idle idle loop clocks the
;6112    ;       VAR with the data on the MC bus.  The MEM.REQ instruction put a 32 bit
;6113    ;       address on the D-BUS from the specified LS address.  The MCT generated
;6114    ;       signal GATE DIR H enables the transceivers on the CPU module through
;6115    ;       the SIGN XTEND CTL PAL to output on the MC BUS.  So the VAR will con-
;6116    ;       tain the 32 bit data  presented on the D-BUS after leaving the idle
;6117    ;       loop.
;6118    ;           The MCT ARBITRATOR PAL will assert CPU GRANT L when it sees MEMORY
;6119    ;       REQUEST H (assuming that the memory is not busy).  This causes the
;6120    ;       memory micro-code to break out of the idle loop.  Back at the ranch
;6121    ;       (the CPU board), CPU GRANT releases the CPU stall (if one  occurred),
;6122    ;       but the CSR will not be loaded with a new u-word immediately.  The
;6123    ;       next state in the MCT u-flow, uses the MF field still in the CSR to
;6124    ;       dispatch to the proper routine.  The dispatch is controlled by the MCT
;6125    ;       u-address bits 7-3 (ADRS 7 L - ADRS 3 L) being fed by the MF bits and
;6126    ;       ADRS 2 - ADRS 0 being fed by the NAD field bits 2-0 of the memory
;6127    ;       controller u-word.  ADRS 8 is also fed by the NAD field (bit 8) unless
;6128    ;       case, ADRS 8 is forced low (asserted).  The VAR is loaded again with
;6129    ;       the data on the CPU D BUS and the u-code jumps to the routine
;6130    ;       specified.
;6131    ;
```

```
;6132   .page   "Micro Tests"
;6133   .toc    " VAR Tests"
;6134
;6135   .toc    "  TEST 1 Write/read VAR Test (M8391  MCT Module and M8390  CPU module)"
;6136   ;++
;6137   ;
;6138   ; Test Description:
;6139   ;
;6140   ;       This test is the first test to use the memory controller for any
;6141   ;       purpose.  The simplest data path is the path to the VAR and back to
;6142   ;       the CPU, but this still involves a lot of untested logic.  The data
;6143   ;       path is as follows:  The CPU issues a MEM.REQ with the MF fields
;6144   ;       indicating READ.MAINT.ADRS (14(H)).  The dispatch in memory occurs as
;6145   ;       described at the beginning of this listing.  After the dispatch, the
;6146   ;       following sequence occurs:  The first u-word at the dispatch address
;6147   ;       sets MEMORY BUSY and enables LVA 09 H through LVA 24 H through two
;6148   ;       quad buffers to the PA bus (PA 09 H through PA 24 H).  It also enables
;6149   ;       four transceivers which put the PA bits and the remaining VAR bits on
;6150   ;       the MC BUS.  The next u-word repeats the previous events (except
;6151   ;       MEMORY BUSY is not asserted).  This u-word also loops on itself
;6152   ;       waiting for CPU GRANT L to go high (to be deasserted).  Back at the
;6153   ;       ranch (the CPU module), DATA REQ H was generated when the MOVE with
;6154   ;       the MDP path=0 (LS and WR gets mem data) was executed.  The signal
;6155   ;       DATA REQ L which generated DATA REQ H also asserts the signal DATA
;6156   ;       RECEIVED H to the MCT module.  DATA RCVD H at the MCT module is
;6157   ;       what allows CPU GRANT to drop, thus releasing the memory u-code from
;6158   ;       its loop.  At this point, the CPU has the data in WR 0 since the MOVE
;6159   ;       reversed the direction of the data transceivers on the CPU module and
;6160   ;       read the data from the MC BUS.  The next u-word in the memory
;6161   ;       controller jumps to the idle loop again.  Data patterns used are
;6162   ;       alternating 1's and 0's, shifting 1's and shifting 0's.  Note that the
;6163   ;       data coming back from the memory during this test will be shifted one
;6164   ;       bit to the right with the LSB sent out coming back as bit 31.
;6165   ;
;6166   ; Assumptions:
;6167   ;
;6168   ;       It is assumed that the CPU diagnostics that run without using the
;6169   ;       memory have run successfully.  It is NOT assumed that the signals on
;6170   ;       the CPU module that go to the MCT module are working correctly.  Thus
;6171   ;       either the CPU or the MCT module could cause a failure.
;6172   ;
;6173   ; Test Steps:
;6174   ;
;6175   ;       1)  Set up error mask, error number, and module number in LS (for
;6176   ;           error printouts) and clear previous error number in LS.
;6177   ;       2)  Load WR 1 with data pattern from LS.  Then load a temporary loc-
;6178   ;           ation in LS with this data pattern.
;6179   ;       3)  Issue a ROR on WR 1 to get expected data in the right position
;6180   ;           (the data comes back shifted right one bit with the LSB of the
;6181   ;           data sent out in bit 31 of the result).
;6182   ;       4)  Issue a MEM.REQ with DT=longword (11) and MF=READ.MAINT.ADRS (14)
;6183   ;           using the temporary location in LS as the source of data.
;6184   ;       5)  Execute a MOV MEM.DATA to WR[0] and check the result.
;6185   ;       6)  Repeat steps 2 through 5 until all data patterns have been tested.
;6186   ;       7)  Repeat steps 2 through 5 using an alternating data pattern and
```

```
;6187    ;           adding a delay (with NOPs) between MEM.REQ and MOV.  This checks
;6188    ;           the CPU GRANT branch logic.
;6189    ;
;6190    ; Errors:
;6191    ;
;6192    ;       Error 1 - READ.MAINT.ADRS failed with data of alternating 1's and 0's.
;6193    ;       Error 2 - READ.MAINT.ADRS failed with data of alternating 0's and 1's.
;6194    ;       Error 3 - READ.MAINT.ADRS failed with data of shifting 1's.
;6195    ;       Error 4 - READ.MAINT.ADRS failed with data of shifting 0's.
;6196    ;       Error 5 - READ.MAINT.ADRS failed with data of alternating 1's and 0's
;6197    ;                 with delay between MEM.REQ and MOV.
;6198    ;
;6199    ; Debug:
;6200    ;
;6201    ;       Two methods of debug will be explained.  If the MCT module is in a
;6202    ;       test station then the MCT can be single stepped making debug easier
;6203    ;       in some cases.  Otherwise only the CPU can be single stepped and this
;6204    ;       method will be explained also.  When single stepping the memory con-
;6205    ;       troller will help in debug, this section will explicitly suggest it.
;6206    ;
;6207    ;       If the CPU is hanging, the signals CPU GRANT L or MEMORY BUSY H are
;6208    ;       probably causing the CPU to stall and are not releasing the CPU.  If
;6209    ;       the CPU is hung at the MEM.REQ instruction, and the memory is at the
;6210    ;       idle loop, check that the signal MEMORY REQUEST H is getting to the
;6211    ;       MCT module at the MCT ARBITRATOR PAL pin 6 (the console will have
;6212    ;       printed a timeout message, so use the console commands to get to this
;6213    ;       instruction: single step or stop on micro match).  If MEMORY REQUEST H
;6214    ;       is low, check the signal EN MEM REF L on the CPU module for a low.
;6215    ;       This signal is controlled by the console.  The signal on pin 5 of the
;6216    ;       'AND' gate should also be low.  If not, check the inputs to the 8 to 1
;6217    ;       MUX for a high on CSR 20 and CSR 19, and a low on CSR 22 and CSR 21.
;6218    ;       If these are correct and the output on pin 6 is high, the MUX is bad.
;6219    ;
;6220    ;       If MEMORY REQUEST H is high at the MCT module's MCT ARBITRATOR PAL,
;6221    ;       check the other inputs for the following: L MSYN H (pin 13) should be
;6222    ;       low, IGN MSYN L (pin 14) should be high (this should be set high by the
;6223    ;       power up u-code), CONT FUNC LAT L (pin 7) should be high, and DATA
;6224    ;       RECEIVED H (pin 9) should be low.  Also check that the enable on pin
;6225    ;       11 is low and that the clock on pin 1 is clocking.  Trace these
;6226    ;       signals back if any are incorrect.  If the inputs are as specified,
;6227    ;       CPU GRANT L should be low, otherwise the PAL is bad.  Unless CPU GRANT
;6228    ;       L can be asserted, the memory u-code will not break out of the idle
;6229    ;       loop.  If the memory is not at the idle loop, it may be that CPU GRANT
;6230    ;       is not getting back to the CPU module.  Check for a float at pin 10 of
;6231    ;       the chip that generates CLOCK STALL L on the CPU module.  If not, the
;6232    ;       best way to debug is to single step the memory controller from power
;6233    ;       up to the idle loop, then execute the MEM.REQ instruction and check
;6234    ;       that CPU GRANT is being asserted low at the CPU module.
;6235    ;
;6236    ;       If the CPU is hung at the MOV instruction that reads data back from
;6237    ;       the memory, MEMORY BUSY H is not going low to release the stall.
;6238    ;       First check MEMORY BUSY H at pin 7 of the chip that generates CLOCK
;6239    ;       STALL L on the CPU module.  If not floating, check the logic that
;6240    ;       generates MEMORY BUSY H on the MCT module.  If MEM BSY H (C38) is
;6241    ;       high, or ERROR L and STALL MBSY L are both low, trace the logic back
```

```
;6242    ;       to the source.  It is possible that the u-sequence logic is failing
;6243    ;       or the memory is stuck in a loop that asserts MEM BSY H.  The best
;6244    ;       way to trace this is to single step the memory controller from the
;6245    ;       idle loop, checking that the u-code is sequencing correctly.
;6246    ;
;6247    ;       Error 1 - This error can be caused by a number of problems.  Since the
;6248    ;       CPU has not hung, some saneness still exists (see description above if
;6249    ;       the CPU has hung).  Single step the CPU to the MEM.REQ instruction.
;6250    ;       The memory controller should already be sending data back on the MC
;6251    ;       BUS.  If the correct data is present at the OCTAL BUS TRNVER's, then
;6252    ;       the problem is probably in the CPU at the MOV instruction that reads
;6253    ;       the data back.  If this is the case, check the enables (pin 19) of the
;6254    ;       transceivers during the MOV instruction for a low.  Also check the
;6255    ;       signal MEM XCVR DIR H for a low.  If these inputs are OK, the
;6256    ;       transceiver is probably bad.  If any of these are in error, check them
;6257    ;       at the outputs of the SIGN XTEND CTL PAL.  The inputs to this PAL
;6258    ;       should be as follows:  DISABLE D-BUS H should be low, MDT 1 H and MDT
;6259    ;       0 H should be high, CSR 22 should be low, CSR 21 should be high, and
;6260    ;       CSR 17 and 18 should be low.  If these inputs are correct, then the
;6261    ;       PAL is bad.  The signals MDT 0 and MDT 1 come from the DATA TYPE CTL
;6262    ;       PAL.  The outputs BUS D D07 H and BUS D D06 H have already been
;6263    ;       checked in the CPU test.  MDT CTL 0 H and MDT CTL 1 H should be
;6264    ;       following these outputs.  If not, the PAL is bad.
;6265    ;         If correct data is not coming back, the problem could be in sending
;6266    ;       the data out, or in the memory controller itself.  During the memory
;6267    ;       idle loop, check the signal GATE DIR H for a high.  If this is incor-
;6268    ;       rect, suspect the 512X8 prom on the MCT module which outputs GATE DIR
;6269    ;       H or its inputs.  The data going out on the MC BUS can be checked
;6270    ;       during the MEM.REQ instruction if the memory controller can be single
;6271    ;       stepped.  Otherwise the signal GATE DIR H will go away after the dis-
;6272    ;       patch.  If MCT single step is available, the enables and dir signals
;6273    ;       going to the transceivers on the CPU module can be checked during the
;6274    ;       MEM.REQ instruction.  These should all be low at that time as long as
;6275    ;       GATE DIR H is high.  These come from the SIGN XTEND CTL PAL and the
;6276    ;       inputs during the MEM.REQ should be as follows:  DISABLE D BUS H
;6277    ;       should be low, GATE DIR H should be high, and CSR 22 H and CSR 21 H
;6278    ;       should be low.  If these are correct, suspect the PAL.  The data
;6279    ;       coming out of the transceivers can also be checked to verify the
;6280    ;       inputs and outputs of the transceiver itself.  The next step, or the
;6281    ;       place to start if MCT single step is not available, is at the output
;6282    ;       of the VAR.  The memory controller will be looping waiting for CPU
;6283    ;       GRANT L to go high.  This should not occur until the MOV instruction
;6284    ;       is issued from the CPU.  If the CPU u-code is stopped at the MEM.REQ
;6285    ;       instruction, the outputs of the VAR can be checked.  If a problem is
;6286    ;       found here, check that VAR MUX SEL H is low and VAR LOAD H is low.
;6287    ;       Also check the enables for a low.  Suspect the VAR PAL or the MC BUS
;6288    ;       if these inputs are OK.  The enable OP ARY ADR L comes from the
;6289    ;       CONTROL PREFETCH PAL and should have been forced low during the idle
;6290    ;       loop.  The inputs CONT FUNC LAT L and CPU GRANT L are both high during
;6291    ;       the idle loop.  This combined with CSR 19 H being high (MEM.REQ
;6292    ;       instruction) should force OP ARY ADR L low regardless of it's previous
;6293    ;       state.  It should not go high unless a DECODE CPU u-instruction is
;6294    ;       executed.  The VAR load can be checked while the memory is in the idle
;6295    ;       loop.  The input VAR LOAD H should be high and VAR MUX SEL H should be
;6296    ;       low.  This combination should latch whatever is on the MC BUS into the
```

```
                ;6297   ;       PAL.
                ;6298   ;         If the VAR outputs are OK, check the inputs to the MCT transceivers
                ;6299   ;       and the signals TB DATA DIR RD L and TB DATA EN+MAINT L for lows.  The
                ;6300   ;       correct data should also be seen on the output (the MC BUS).  If bits
                ;6301   ;       BUS MC 08 through 23 are in error, check the quad buffers that output
                ;6302   ;       PA 09 through PA 24.  The enable ADDR PH L comes from the MISC CONTROL
                ;6303   ;       PAL and should be low.  The inputs to this PAL should be a low on SPF1
                ;6304   ;       H and SPF2 H and a high on SPF0 H.
                ;6305   ;         Another possibility for failure exists within the u-sequencer logic.
                ;6306   ;       If MCT single step is not available, it will be difficult to debug
                ;6307   ;       this logic.  If the memory is looping at the correct spot before the
                ;6308   ;       CPU issues a MOV instruction, the u-sequencer is probably OK for this
                ;6309   ;       test.
                ;6310   ;
                ;6311   ;       Errors 2 to 4 - If this is the first failure, suspect the MCT module.
                ;6312   ;       The VAR PALS, the buffer that outputs PA 09 through PA 24, and the MCT
                ;6313   ;       transceivers are the most likely suspects.
                ;6314   ;
                ;6315   ;       Error 5 - This error indicates a problem with the CPU GRANT branch
                ;6316   ;       logic.  The memory controller is supposed to loop waiting for data
                ;6317   ;       received to clear CPU GRANT.  If this logic fails to cause a loop, the
                ;6318   ;       memory controller will disable the output on the MC BUS before the CPU
                ;6319   ;       has time to read it.  Single step the CPU to the MEM.REQ instruction
                ;6320   ;       and check the signal DATA RCVD H for a low at the MCT ARBITRATOR PAL.
                ;6321   ;       If OK, check that CPU GRANT L is still low at the branch logic (BEN 1
                ;6322   ;       MUX).
                ;6323   ;
                ;6324   ;--
                ;6325
                ;6326   T.1:
                ;6327
U 0700, B65E,15 ;6328           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;6329                                           ; STATUS WORD
U 0701, 3E80,15 ;6330           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;6331                                           ; BIT 15 INDICATES START OF TEST TO
                ;6332                                           ; CONSOLE
U 0702, 10E0,15 ;6333           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;6334   WAIT.T1.0:
U 0703, 0870,34 ;6335           JMP [WAIT.T1.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                ;6336                                           ; RESPOND
                ;6337
U 0704, 0A1A,AC ;6338           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
U 0705, 4742,15 ;6339           BIS LS[CPU] TO WR[0]            ;ADD CPU TO MODULE CODE
U 0706, 3E8C,15 ;6340           MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                ;6341                                           ; CPU AND MCT BOARDS
                ;6342   LOOP.T1.1:
U 0707, B69A,95 ;6343           MOV LS[ALTER.EVEN] TO WR[1]     ;EXPECTED DATA (DATA IS SHIFTED 1 BIT
                ;6344                                           ; LEFT)
U 0708, 9D98,75 ;6345           MEM.REQ[READ.MAINT.ADRS] ADRS[ALTER.ODD] DT[LONG]
                ;6346                                           ; WRITE ALTERNATING DATA PATTERN IN VAR
U 0709, 3022,15 ;6347           MOV MEM.DATA TO WR[0]           ;READ PATTERN BACK TO WR[0]
U 070A, 0869,3C ;6348           JSR [CHECK.RESULT]              ;CHECK THE ANSWER
U 070B, 8870,74 ;6349           JMP [LOOP.T1.1]                 ;ERROR LOOP IF ENABLED
                ;6350
U 070C, FF82,15 ;6351           INC LS[ERROR.NUMBER]            ;ERROR NUMBER 2
```

```
                    ;6352   LOOP.T1.2:
U 070D, 3698,95     ;6353           MOV LS[ALTER.ODD] TO WR[1]       ;NEXT EXPECTED DATA PATTERN
U 070E, 1D9A,75     ;6354           MEM.REQ[READ.MAINT.ADRS] ADRS[ALTER.EVEN] DT[LONG]
                    ;6355                                            ; WRITE ALTERNATING DATA PATTERN IN VAR
U 070F, 3022,15     ;6356           MOV MEM.DATA TO WR[0]            ;READ PATTERN BACK TO WR 0
U 0710, 0869,3C     ;6357           JSR [CHECK.RESULT]               ;CHECK THE ANSWER
U 0711, 8870,D4     ;6358           JMP [LOOP.T1.2]                  ;ERROR LOOP IF ENABLED
                    ;6359
U 0712, FF82,15     ;6360           INC LS[ERROR.NUMBER]             ;ERROR NUMBER 3
U 0713, E5F8,15     ;6361           CLR LS[OS]                       ;CLEAR THE OS REG (USED FOR INDEXING)
                    ;6362   LOOP.T1.3:
U 0714, B6F6,95     ;6363           MOV LS[SHIFT.OS(4-0)] TO WR[1]   ;LOAD DATA PATTERN INTO WR[1]
U 0715, A340,95     ;6364           ROR WR[1]                        ;SHIFT DATA RIGHT TO CORRECT FOR HARD-
                    ;6365                                            ; WARE SHIFT ON MCT
U 0716, 1DF6,75     ;6366           MEM.REQ[READ.MAINT.ADRS] ADRS[SHIFT.OS(4-0)] DT[LONG]
                    ;6367                                            ; WRITE SHIFTING ONES PATTERN IN VAR
U 0717, 3022,15     ;6368           MOV MEM.DATA TO WR[0]            ;GET RESULT
U 0718, 0869,3C     ;6369           JSR [CHECK.RESULT]               ;AND CHECK
U 0719, 0871,44     ;6370           JMP [LOOP.T1.3]                  ;ERROR LOOP IF ENABLED
                    ;6371
U 071A, 7FF8,15     ;6372           INC LS[OS]                       ;INCREMENT INDEX FOR NEXT SHIFT PATTERN
U 071B, 36F9,15     ;6373           MOV LS[OS] TO WR[2]              ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                    ;6374           BIT LS[BIT5] WITH WR[2],         ;SEE IF OS HAS INCREMENTED TO 20(H)
U 071C, D94B,35     ;6375            DT(LONG)&SET.ALU.CC             ; AND SET UP CONDITION CODES
U 071D, 8871,49     ;6376           JMP.IF[BITS.CLR] TO [LOOP.T1.3] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                    ;6377
U 071E, FF82,15     ;6378           INC LS[ERROR.NUMBER]             ;ERROR NUMBER 4
U 071F, E5F8,15     ;6379           CLR LS[OS]                       ;CLEAR THE OS REG (USED FOR INDEXING)
                    ;6380   LOOP.T1.4:
U 0720, DFF6,95     ;6381           MCOM LS[SHIFT.OS(4-0)] TO WR[1] ;LOAD SHIFTING 0'S DATA PATTERN INTO
                    ;6382                                            ; WR[1]
U 0721, BE10,95     ;6383           MOV WR[1] TO LS[T8]              ;SAVE DATA PATTERN TO WRITE IN TEMP LS
                    ;6384                                            ; LOCATION
U 0722, A340,95     ;6385           ROR WR[1]                        ;SHIFT DATA RIGHT TO CORRECT FOR HARD-
                    ;6386                                            ; WARE SHIFT ON MCT
U 0723, 9D10,75     ;6387           MEM.REQ[READ.MAINT.ADRS] ADRS[T8] DT[LONG]
                    ;6388                                            ; WRITE SHIFTING ZEROS PATTERN IN VAR
U 0724, 3022,15     ;6389           MOV MEM.DATA TO WR[0]            ;GET RESULT
U 0725, 0869,3C     ;6390           JSR [CHECK.RESULT]               ;AND CHECK
U 0726, 8872,04     ;6391           JMP [LOOP.T1.4]                  ;ERROR LOOP IF ENABLED
                    ;6392
U 0727, 7FF8,15     ;6393           INC LS[OS]                       ;INCREMENT INDEX FOR NEXT SHIFT PATTERN
U 0728, 36F9,15     ;6394           MOV LS[OS] TO WR[2]              ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                    ;6395           BIT LS[BIT5] WITH WR[2],         ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0729, D94B,35     ;6396            DT(LONG)&SET.ALU.CC             ; AND SET UP CONDITION CODES
U 072A, 0872,09     ;6397           JMP.IF[BITS.CLR] TO [LOOP.T1.4] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                    ;6398
U 072B, FF82,15     ;6399           INC LS[ERROR.NUMBER]             ;ERROR NUMBER 5
                    ;6400   LOOP.T1.5:
U 072C, B69A,95     ;6401           MOV LS[ALTER.EVEN] TO WR[1]      ;EXPECTED DATA (DATA IS SHIFTED 1 BIT
                    ;6402                                            ; LEFT)
U 072D, 9D98,75     ;6403           MEM.REQ[READ.MAINT.ADRS] ADRS[ALTER.ODD] DT[LONG]
                    ;6404                                            ; WRITE ALTERNATING DATA PATTERN IN VAR
U 072E, 0A1B,4C     ;6405           JSR [NOP.DELAY]                  ;EXECUTE 5 CYCLE DELAY TO CHECK MEMORY
                    ;6406                                            ; CPU GRANT BRANCH
```

```
U 072F, 3022,15 ;6407            MOV MEM.DATA TO WR[0]          ;READ PATTERN BACK TO WR[0]
U 0730, 0869,3C ;6408            JSR [CHECK.RESULT]             ;CHECK THE ANSWER
U 0731, 8872,C4 ;6409            JMP [LOOP.T1.5]                ;ERROR LOOP IF ENABLED
                ;6410   END.T1:
```

```
;6411   .PAGE   "  TEST 2 Inc VAR and Stall On Mem Busy Test (M8390 CPU and M8391  MCT Module)"
;6412   ;++
;6413   ;
;6414   ; Test Description:
;6415   ;
;6416   ;       This test checks the increment VAR function and assures that the CPU
;6417   ;       stalls when a read is executed and the memory is busy.  The data path
;6418   ;       is as follows:  The CPU issues a MEM.REQ with the MF field=RD.MAINT.
;6419   ;       VAR .INC (08H).  The dispatch to memory occurs as described in the
;6420   ;       beginning of the listing.  At the dispatch address, memory busy is set
;6421   ;       and address physical is enabled.  The next u-cycle does an increment
;6422   ;       on the VAR and keeps memory busy asserted.  This cycle is followed by
;6423   ;       four (4) memory cycles which simply keep memory busy asserted.  This
;6424   ;       delay is necessary to check that the CPU stall on memory busy is
;6425   ;       working correctly.  If the CPU fails to stall, the CPU will miss the
;6426   ;       data sent back and the failure will be detected.  After the 4 cycle
;6427   ;       delay, the VAR is read by enabling the VAR through the address
;6428   ;       physical quad buffers to the BUS MC quad buffers and octal bus
;6429   ;       transcievers and placed on the MC BUS.  Also memory busy is dropped
;6430   ;       and the memory U-code loops waiting for CPUG to drop.  A MOV
;6431   ;       instruction at the CPU has been stalled until memory busy went away
;6432   ;       and now will read the data on the MC bus and verify it.  The memory
;6433   ;       returns to idle after checking for another CPU request.
;6434   ;         The VAR increments by 4 (LVA 00 and LVA 01 are unchanged).  Also LVA
;6435   ;       30 and LVA 31 are unchanged.  The data patterns used are as follows:
;6436   ;       All 0's, all 1's, shifting 1's, and shifting 0's with an additional
;6437   ;       bit cleared (shifting 0's is repeated 32 times, and each time a
;6438   ;       different bit is cleared in addition to the shifting bit, starting at
;6439   ;       bit 0 up to bit 31).  Before each pattern is tested, the compliment of
;6440   ;       the current pattern is written in the VAR.
;6441   ;
;6442   ; Assumptions:
;6443   ;
;6444   ;       It is assummed that the previous VAR test has run successfully.  There
;6445   ;       is a small peice of logic on the CPU module (the stall on memory busy)
;6446   ;       that is tested for the first time here.
;6447   ;
;6448   ; Test Steps:
;6449   ;
;6450   ;       1)  Set up error mask, error number, and module number in LS (for
;6451   ;           error printouts) and clear previous error number in LS.
;6452   ;       2)  Clear WR 0 and move this value to a temporary LS location.
;6453   ;       3)  Load WR 1 with a 2(H) as expected value (result is shifted right 1
;6454   ;           bit when read back).
;6455   ;       4)  Write complement data in VAR, then issue MEM.REQ with DT=LONGWORD
;6456   ;           and MF=RD.MAINT.VAR.INC using the temporary location in LS as data
;6457   ;           (all 0's).
;6458   ;       5)  Issue a MOV MEM.DATA to WR[0] and check result.
;6459   ;       6)  Repeat steps 2 through 5 using data of all 1's.  Result should be
;6460   ;           all 0's w    1's in bits 31, 30, 29, and 0 after shift.
;6461   ;       7)  Repeat step  2 through 5 using data of shifting 1's.
;6462   ;       8)  Repeat steps 2 through 5 using data of shifting 0's and clearing
;6463   ;           an additional bit in succession.
;6464   ;
;6465   ; Errors:
```

```
;6466   ;
;6467   ;       Error 1 - INC VAR failed with starting data of all 0's.
;6468   ;       Error 2 - INC VAR failed with starting data of all 1's.
;6469   ;       Error 3 - INC VAR failed with shifting 1 pattern.
;6470   ;       Error 4 - INC VAR failed with shifting 0 pattern with additional bit
;6471   ;                 cleared.
;6472   ;
;6473   ; Debug:
;6474   ;
;6475   ;       Two methods of debug will be explained.  If the MCT module is in a
;6476   ;       test station then the MCT can be single stepped making debug easier
;6477   ;       in some cases.  Otherwise only the CPU can be single stepped and this
;6478   ;       method will be explained also.  When single stepping the memory con-
;6479   ;       troller will help in debug, this section will explicitly suggest it.
;6480   ;
;6481   ;       Error 1 - A failure here could be caused by either the memory VAR in-
;6482   ;       crement logic or the CPU stall logic.  If the actual result is all
;6483   ;       ones the CPU stall logic should be suspected first.  First single step
;6484   ;       the CPU to the MOV MEM.DATA TO WR[0] instruction and check the signal
;6485   ;       DATA REQ H going to the clock stall logic on the CPU (pin 2).  This
;6486   ;       should be high during the MOV instruction.  If this is OK, the signal
;6487   ;       MEMORY BUSY should be checked.  If the MCT can be single stepped, step
;6488   ;       the memory u-code to the first instruction after the dispatch address
;6489   ;       (the one that sets address physical and memory busy) the check MEMORY
;6490   ;       BUSY H at the CPU clock stall logic (pin 1) for a high.  If the CPU is
;6491   ;       at the MOV instruction, DATA REQ H and MEMORY BUSY H should produce a
;6492   ;       low on CLOCK STALL L.  If MEMORY BUSY H is low, check the output at
;6493   ;       the MCT module.  It can be traced back to C38 of the MCT u-code.  If
;6494   ;       MCT single step is not available, loop on error and check the signals
;6495   ;       while looping.
;6496   ;         If the result is not all 1's, the problem is probably at the MCT
;6497   ;       module.  If MCT single step is available, check the inputs to the VAR
;6498   ;       PALs during the memory u-code that increments VAR for the following:
;6499   ;       VAR MUX SEL H should be high and VAR LOAD H should be low.  Both of
;6500   ;       these signals can be traced back to the MCT u-store proms (C20 and
;6501   ;       C21).  If these signals are OK, check that pin 19 of the lowest order
;6502   ;       VAR PAL is tied high.  If not, no increment can occur.  The CIN to the
;6503   ;       other VAR PALs (pin 19)should all be low for this data pattern.  The
;6504   ;       VAR PAL that is causing the problem can be identified by determining
;6505   ;       the first (lowest order) bit to fail.  Remember that the result is
;6506   ;       shifted 1 bit to left.  That is if bit 9 of the result is wrong, then
;6507   ;       VAR bit 8 is incorrect.  If CIN and VAR control bits are OK, suspect
;6508   ;       the VAR PAL assocciated with the failing bit.  If MCT single step is
;6509   ;       not available, the signals will have to be checked while looping on
;6510   ;       error.
;6511   ;
;6512   ;       Error 2 - If this is the first error, suspect the VAR PAL itself.  The
;6513   ;       failing PAL can be determined by identifying the first failing bit
;6514   ;       (lowest order bit).  Remember that the result is shifted 1 bit to the
;6515   ;       left, so if bit 9 of the result is failing, then bit 8 of the VAR PAL
;6516   ;       is causing the failure.  With this data pattern, all CIN signals (pin
;6517   ;       19) should be high.  If not, suspect the preceeding VAR PAL.
;6518   ;
;6519   ;       Error 3 and 4 - These errors indicate problems with the VAR PAL it-
;6520   ;       self.  Determine the failing VAR bit as described in error 2 and
```

```
                ;6521   ;       suspect that PAL.
                ;6522   ;
                ;6523   ;--
                ;6524
                ;6525   T.2:
                ;6526
U 0732, B65E,15 ;6527           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;6528                                           ; STATUS WORD
U 0733, 3E80,15 ;6529           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;6530                                           ; BIT 15 INDICATES START OF TEST TO
                ;6531                                           ; CONSOLE
U 0734, 10E0,15 ;6532           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;6533   WAIT.T2.0:
U 0735, 0873,54 ;6534           JMP [WAIT.T2.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                ;6535                                           ; RESPOND
                ;6536
U 0736, 0A1A,AC ;6537           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
U 0737, 4742,15 ;6538           BIS LS[CPU] TO WR[0]            ;ADD CPU TO MODULE CODE
U 0738, 3E8C,15 ;6539           MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                ;6540                                           ; CPU AND MCT BOARDS
                ;6541   LOOP.T2.1:
U 0739, B642,95 ;6542           MOV LS[#2] TO WR[1]             ;EXPECTED DATA (0+4=4 SHIFTED RIGHT=2)
U 073A, 9D9E,75 ;6543           MEM.REQ[READ.MAINT.ADRS] ADRS[ONES] DT[LONG]
                ;6544                                           ;SET UP TO WRITE COMPLEMENT DATA IN VAR
U 073B, 3022,15 ;6545           MOV MEM.DATA TO WR[0]           ;EXECUTE MOVE TO COMPLETE CYCLE BUT DO
                ;6546                                           ; NOT CHECK
U 073C, 9A9D,F5 ;6547           MEM.REQ[READ.MAINT.VAR.INC] ADRS[#0] DT[LONG]
                ;6548                                           ;SEND DATA OF 0 TO VAR TO BE INCREMENTED
U 073D, 3022,15 ;6549           MOV MEM.DATA TO WR[0]           ;GET RESULT
U 073E, 0869,3C ;6550           JSR [CHECK.RESULT]              ;CHECK THE ANSWER
U 073F, 0873,94 ;6551           JMP [LOOP.T2.1]                 ;LOOP ON ERROR IF ENABLED
                ;6552
U 0740, FF82,15 ;6553           INC LS[ERROR.NUMBER]            ;ERROR 2
U 0741, 2F87,95 ;6554           CLR WR[3]                       ;CLR EXPECTED RESULT
U 0742, 477F,95 ;6555           BIS LS[BIT31] TO WR[3]          ;SET BIT 31
U 0743, C77D,95 ;6556           BIS LS[BIT30] TO WR[3]          ;SET BIT 30
U 0744, C77B,95 ;6557           BIS LS[BIT29] TO WR[3]          ;SET BIT 29
U 0745, C741,95 ;6558           BIS LS[BIT0] TO WR[3]           ;SET BIT 0.  NOW HAVE EXPECTED RESULT
                ;6559   LOOP.T2.2:
U 0746, 2006,95 ;6560           MOV WR[3] TO WR[1]              ;PUT EXPECTED RESULT IN WR 1
U 0747, 1D9C,75 ;6561           MEM.REQ[READ.MAINT.ADRS] ADRS[ZERO] DT[LONG]
                ;6562                                           ;SET UP TO WRITE COMPLEMENT DATA IN VAR
U 0748, 3022,15 ;6563           MOV MEM.DATA TO WR[0]           ;EXECUTE MOVE TO COMPLETE CYCLE BUT DO
                ;6564                                           ; NOT CHECK
U 0749, 1A9F,F5 ;6565           MEM.REQ[READ.MAINT.VAR.INC] ADRS[ONES] DT[LONG]
                ;6566                                           ;SEND DATA OF ALL ONES TO VAR TO BE INC-
                ;6567                                           ; REMENTED
U 074A, 3022,15 ;6568           MOV MEM.DATA TO WR[0]           ;GET RESULT
U 074B, 0869,3C ;6569           JSR [CHECK.RESULT]              ;CHECK THE RESULT
U 074C, 8874,64 ;6570           JMP [LOOP.T2.2]                 ;ERROR LOOP IF ENABLED
                ;6571
U 074D, FF82,15 ;6572           INC LS[ERROR.NUMBER]            ;ERROR 3
U 074E, E5F8,15 ;6573           CLR LS[OS]                      ;CLEAR INDEX
                ;6574   REPEAT.T2.3:
U 074F, 36F7,95 ;6575           MOV LS[SHIFT.OS(4-0)] TO WR[3]  ;GET DATA PATTERN
```

```
U 0750, 7815,95 ;6576          MCOM WR[3] TO LS[T10]           ;PUT COMPLEMENT DATA IN LS
U 0751, C045,95 ;6577          ADD LS[#4] TO WR[3]             ;ADD 4 TO DATA PATTERN (LS INCREMENTS
                ;6578                                          ; BY 4)
U 0752, 2341,95 ;6579          ROR WR[3]                       ;SHIFT RIGHT LIKE HARDWARE DOES
                ;6580    LOOP.T2.3:
U 0753, 2006,95 ;6581          MOV WR[3] TO WR[1]              ;SET UP EXPECTED DATA
U 0754, 1D14,75 ;6582          MEM.REQ[READ.MAINT.ADRS] ADRS[T10] DT[LONG]
                ;6583                                          ;SET UP TO WRITE COMPLEMENT DATA IN VAR
U 0755, 3022,15 ;6584          MOV MEM.DATA TO WR[0]           ;EXECUTE MOVE TO COMPLETE CYCLE BUT DO
                ;6585                                          ; NOT CHECK
U 0756, 9AF7,F5 ;6586          MEM.REQ[READ.MAINT.VAR.INC] ADRS[SHIFT.OS(4-0)] DT[LONG]
                ;6587                                          ; SEND DATA PATTERN TO VAR
U 0757, 3022,15 ;6588          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0758, 0869,3C ;6589          JSR [CHECK.RESULT]              ;CHECK THE RESULT
U 0759, 0875,34 ;6590          JMP [LOOP.T2.3]                 ;ERROR LOOP IF ENABLED
                ;6591
U 075A, 7FF8,15 ;6592          INC LS[OS]                      ;INCREMENT INDEX
U 075B, 36F9,15 ;6593          MOV LS[OS] TO WR[2]             ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                ;6594          BIT LS[BIT5] WITH WR[2],        ;SEE IF OS HAS INCREMENTED TO 20(H)
U 075C, D94B,35 ;6595           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 075D, 0874,F9 ;6596          JMP.IF[BITS.CLR] TO [REPEAT.T2.3] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                ;6597
U 075E, FF82,15 ;6598          INC LS[ERROR.NUMBER]            ;ERROR NUMBER 4
U 075F, E5F8,15 ;6599          CLR LS[OS]                      ;CLEAR INDEX
                ;6600
U 0760, DF41,95 ;6601          MCOM LS[BIT0] TO WR[3]          ;WR[3] NOW CONTAINS A SHIFTING 0 PATTERN
                ;6602                                          ; (FFFFFFFE)
U 0761, 3E11,95 ;6603          MOV WR[3] TO LS[T8]             ;STORE SHIFTING PATTERN IN TEMPORARY LS
                ;6604                                          ; LOCATION.
                ;6605    REPEAT.T2.4:
U 0762, C5F7,95 ;6606          BIC LS[SHIFT.OS(4-0)] TO WR[3]  ;CLEAR AN ADDITIONAL BIT
U 0763, BE13,95 ;6607          MOV WR[3] TO LS[T9]             ;LS T9 WILL BE USED AS SOURCE OF DATA
                ;6608                                          ; FOR MEM.REQ
U 0764, 7815,95 ;6609          MCOM WR[3] TO LS[T10]           ;PUT COMPLEMENT DATA IN LS
                ;6610
U 0765, 2F85,15 ;6611          CLR WR[2]                       ;USE WR[2] TO STORE STATE OF BITS 30
                ;6612                                          ; AND 31
                ;6613          BIT LS[BIT30] WITH WR[3],       ;SEE IF BIT 30 IS SET
U 0766, 597D,B5 ;6614           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 0767, 5B00,09 ;6615          SKIP.IF[BITS.CLR]               ;SKIP NEXT INSTRUCTION IF BIT 30 CLEAR
U 0768, 477D,15 ;6616          BIS LS[BIT30] TO WR[2]          ;ELSE SET BIT IN WR 2
                ;6617
                ;6618          BIT LS[BIT31] WITH WR[3],       ;SEE IF BIT 31 IS SET
U 0769, D97F,B5 ;6619           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 076A, 5B00,09 ;6620          SKIP.IF[BITS.CLR]               ;SKIP NEXT INSTRUCTION IF BIT 31 CLEAR
U 076B, C77F,15 ;6621          BIS LS[BIT31] TO WR[2]          ;ELSE SET BIT IN WR 2
                ;6622
U 076C, C045,95 ;6623          ADD LS[#4] TO WR[3]             ;ADD 4 TO DATA PATTERN (LS INCREMENTS
                ;6624                                          ; BY 4)
U 076D, 457D,95 ;6625          BIC LS[BIT30] TO WR[3]          ;CLEAR BIT 30
U 076E, C57F,95 ;6626          BIC LS[BIT31] TO WR[3]          ;AND BIT 31
U 076F, AEC5,95 ;6627          BIS WR[2] TO WR[3]              ;SET UP BITS 30 AND 31 (NOT ALTERED IN
                ;6628                                          ; MCT BY INCREMENT)
U 0770, 2341,95 ;6629          ROR WR[3]                       ;SHIFT RIGHT LIKE HARDWARE DOES
                ;6630    LOOP.T2.4:
```

```
U 0771, 2006,95 ;6631          MOV WR[3] TO WR[1]                ;SET UP EXPECTED DATA
U 0772, 1D14,75 ;6632          MEM.REQ[READ.MAINT.ADRS] ADRS[T10] DT[LONG]
                ;6633                                            ;SET UP TO WRITE COMPLEMENT DATA IN VAR
U 0773, 3022,15 ;6634          MOV MEM.DATA TO WR[0]             ;EXECUTE MOVE TO COMPLETE CYCLE BUT DO
                ;6635                                            ; NOT CHECK
U 0774, 9A13,F5 ;6636          MEM.REQ[READ.MAINT.VAR.INC] ADRS[T9] DT[LONG]
                ;6637                                            ; SEND DATA PATTERN TO VAR
U 0775, 3022,15 ;6638          MOV MEM.DATA TO WR[0]             ;GET RESULT
U 0776, 0869,3C ;6639          JSR [CHECK.RESULT]                ;CHECK THE RESULT
U 0777, 0877,14 ;6640          JMP [LOOP.T2.4]                   ;ERROR LOOP IF ENABLED
                ;6641
U 0778, B611,95 ;6642          MOV LS[T8] TO WR[3]               ;GET OLD SHIFTING 0 PATTERN
U 0779, A3C1,95 ;6643          ROL WR[3]                         ;CREATE NEXT SHIFTING 0 PATTERN
U 077A, 3E11,95 ;6644          MOV WR[3] TO LS[T8]               ;SAVE NEW PATTERN IN LS T8
                ;6645          BIT LS[BIT0] WITH WR[3],          ;SEE IF DONE WITH SHIFT PATTERN.  BIT 0
                ;6646                                            ; WILL BE CLEAR WHEN DONE
U 077B, 5941,B5 ;6647           DT(LONG)&SET.ALU.CC              ; SET CONDITION CODES
U 077C, 8876,21 ;6648          JMP.IF[BIT.SET] TO [REPEAT.T2.4]
                ;6649                                            ;REPEAT SHIFT PATTERN IF BIT 0 STILL SET
U 077D, 7FF8,15 ;6650          INC LS[OS]                        ;ELSE INCREMENT INDEX TO BIT CLEAR A
                ;6651                                            ; DIFFERENT BIT
U 077E, B6F7,15 ;6652          MOV LS[SHIFT.OS(4-0)] TO WR[2]    ;GOT TO CHECK BIC PATTERN TO SEE IF
                ;6653                                            ; TEST DONE
                ;6654          BIT LS[BIT0] WITH WR[2],          ;DONE WHEN BIT 0 IS SET (JUST FINISHED
                ;6655                                            ; BIC ON BIT 31 PATTERN)
U 077F, D941,35 ;6656           DT(LONG)&SET.ALU.CC              ; SET CONDITION CODES
U 0780, 0876,29 ;6657          JMP.IF[BITS.CLR] TO [REPEAT.T2.4]
                ;6658                                            ;START SHIFT PATTERN ANEW WITH DIFFERENT
                ;6659                                            ; ADDITIONAL BIT BEING CLEARED UNTIL DONE
                ;6660   END.T2:
```

```
;6661   .PAGE   " Basic CSR 1 Test and Limited Branch tests"
;6662
;6663   .toc    "  TEST 3 CSR 1 (partial) Read/Write Test (M8391 MCT and M8390 CPU Module)"
;6664   ;++
;6665   ;
;6666   ; Test Description:
;6667   ;
;6668   ;       This test checks bits 29-25 of CSR 1 for read/write capability.
;6669   ;       Other bits in CSR 1 will be checked in subsequent tests.  This test
;6670   ;       uses the WRITE.CSR and READ.CSR entry points associated with the
;6671   ;       MEM.REQ CPU u-instruction.  The data path is as follows:  The CPU
;6672   ;       issues a MEM.REQ u-instruction with the MF field set to WRITE.CSR
;6673   ;       (17H).  The dispatch to memory occurs as described at the beginning of
;6674   ;       this listing.  The address sent to the memory controller has bits 3
;6675   ;       and 2 set to 01(B).  This is latched into the VAR and will be used to
;6676   ;       select CSR 1.  After the dispatch, the following sequence occurs: At
;6677   ;       the dispatch address, memory busy is set and the transceivers from the
;6678   ;       CPU are enabled.  The next memory cycle keeps the CPU transcievers
;6679   ;       enabled, keeps memory busy set, and enables a clear signal to CSR 1
;6680   ;       error flags.  This u-word will loop waiting for a data request from
;6681   ;       the CPU.  Data request is generated via a CPU MOV instruction with
;6682   ;       memory as the destination.  The CPU sends the data pattern to be
;6683   ;       tested with a MOV instruction, and when DATA REQ is received at the
;6684   ;       memory controller, the memory u-code goes on to the next cycle.  The
;6685   ;       data from the CPU is on the MC BUS at this time and this u-cycle keeps
;6686   ;       memory busy asserted to prevent the CPU from going on to the next
;6687   ;       u-instruction (the CPU is stalled until memory busy goes away).  In
;6688   ;       addition, this cycle does the write to the CSR by asserting WR CSR L.
;6689   ;       This signal asserts WR CSR 1 H since LVA 03 and LVA 02 is 01(B).  WR
;6690   ;       CSR 1 is generated by the CONTROL PREFETCH PAL.  The data on BUS MC
;6691   ;       D29 through BUS MC D25 is now written into CSR 1 bits 29 through 25.
;6692   ;       The next u-cycle deasserts memory busy, prepares for another CPU
;6693   ;       request, and goes to the idle loop.  Deasserting memory busy releases
;6694   ;       the CPU stall.  A read CSR is now executed.  A MEM.REQ is made with
;6695   ;       the MF field set to READ.CSR (16H).  The address sent over is the same
;6696   ;       as for the write above.  At the dispatch address, memory busy is set
;6697   ;       to stall the CPU when the CPU MOV instruction is issued to read the
;6698   ;       data back.  the next memory u-cycle keeps memory busy asserted and
;6699   ;       asserts RD CSR L.  This combined with LVA 2 and 3 asserts RD CSR 1 L
;6700   ;       which enables the data from CSR 1 onto the MC BUS.  The next mem
;6701   ;       u-cycle drops memory busy and loops waiting for CPU DATA RCVD H to
;6702   ;       clear CPU GRANT.  When CPU GRANT is cleared, the CSR data has been
;6703   ;       read by the CPU MOV instruction and can be checked.  The mem u-code
;6704   ;       goes on to prepare for another CPU request, and then returns to the
;6705   ;       idle loop.  Data patterns are all 1's, all 0's and shifting 1's.
;6706   ;
;6707   ; Assumptions:
;6708   ;
;6709   ;       It is assumed that the VAR test has run successfully.  This test uses
;6710   ;       one new signal from the CPU module, DATA REQ H, which could cause a
;6711   ;       failure.  The signal has been tested internally but the output finger
;6712   ;       has not been used before.
;6713   ;
;6714   ; Test Steps:
;6715   ;
```

```
;6716    ;       1)  Set up error mask, error number, and module number in LS (for
;6717    ;           error printouts) and clear previous error number in LS.
;6718    ;       2)  Load WR 1 with the data pattern to be used.
;6719    ;       3)  Load bits 29-25 of WR 0 with ones (the complement of the
;6720    ;           first data pattern).  Execute MEM.REQ with DT=LONGWORD and MF=
;6721    ;           WRITE.CSR.  Use the LS location containing a 4(H) as the ad-
;6722    ;           dress.
;6723    ;       4)  Execute several delays (NOPs) to assure that the memory controller
;6724    ;           is waiting for DATA REQUEST.  Then execute MOV from LS to memory
;6725    ;           to load CSR 1 with data.
;6726    ;       5)  Execute MEM.REQ with DT=LONGWORD and MF=READ.CSR.  Use the LS
;6727    ;           location containing the constant 4(H) as the virtual address (this
;6728    ;           selects CSR 1).
;6729    ;       6)  Execute MOV from mem to WR[0] to read CSR 1 and check result.
;6730    ;       7)  Repeat steps 2 through 6 for each data pattern.
;6731    ;
;6732    ; Errors:
;6733    ;
;6734    ;       Error 1 - Write or read of CSR 1 failed with data of all 0's.
;6735    ;       Error 2 - Write or read of CSR 1 failed with data of all 1's.
;6736    ;       Error 3 - Write or read of CSR 1 failed with data of shifting 1's.
;6737    ;
;6738    ; Debug:
;6739    ;
;6740    ;       Two methods of debug will be explained.  If the MCT module is in a
;6741    ;       test station then the MCT can be single stepped making debug easier
;6742    ;       in some cases.  Otherwise only the CPU can be single stepped and this
;6743    ;       method will be explained also.  When single stepping the memory con-
;6744    ;       troller will help in debug, this section will explicitly suggest it.
;6745    ;
;6746    ;       Error 1 - This error indicates a problem with either reading or
;6747    ;       writing CSR 1 bits 19-15.  If one or two bits are failing check the
;6748    ;       inputs BUS MC D29 through BUS MC D25 on the CSR 1 latch for floats.
;6749    ;       Also check inputs to the quad buffer used on the read for floats.  Any
;6750    ;       open etches at these points would cause a 1 instead of a 0.  If no
;6751    ;       floats are found single step the CPU up to the MEM.REQ instruction.
;6752    ;       The memory u-code should be looping on the instruction that looks for
;6753    ;       CPU DATA REQUEST to become asserted (this is the first instruction
;6754    ;       after the dispatch u-word).
;6755    ;         If the memory is not looping on that u-word, there is a problem with
;6756    ;       the sequence logic or the branching logic.  Check the signal L CPU DR
;6757    ;       H at the mux that supplies ADDR 1 L (the u-address).  It should be
;6758    ;       low.  If not, trace the signal back to the latch and finally to the
;6759    ;       finger of the MCT module.  Also check the clock and the enable on this
;6760    ;       latch.  If DATA REQ H is incorrect, check the finger at the CPU board.
;6761    ;       If L CPU DR H is OK, the easiest way to trace the sequence is by
;6762    ;       single stepping the memory controller.  Check for a 6(H) on the select
;6763    ;       lines of the branch mux (BEN 1) and for a high on the ADDR 1 L output.
;6764    ;         If the memory is looping on the correct u-word, check the inputs to
;6765    ;       the CSR 1 latch again for the correct data.  Single step the CPU to
;6766    ;       the MOV WR[1] to MEM instruction and check that the memory u-code gets
;6767    ;       out of it's loop and returns to idle.  If not, check L CPU DR H as
;6768    ;       described above for a high.  If the memory u-code does advance, check
;6769    ;       the signal WR CSR 1 H to the CSR 1 latch for a low to high transition.
;6770    ;       Also check that CLR CSR L remains high (this is easiest done with MCT
```

```
                ;6771   ;       single step, but can be checked in an error loop).  WR CSR 1 H comes
                ;6772   ;       from the CONTROL PREFETCH PAL and is produced by a low on WR CSR L, a
                ;6773   ;       low on LVA 03 H and a high on LVA 02 H.
                ;6774   ;         The output of the CSR 1 latch can be checked after the memory re-
                ;6775   ;       turns to idle.  Also check the inputs into the quad buffer.
                ;6776   ;         The read portion can be checked by single stepping the CPU to the
                ;6777   ;       2nd MEM.REQ (i.e. the one with MF=READ.CSR).  The memory u-code will
                ;6778   ;       then proceed to a loop, waiting for DATA RCVD H from the CPU to
                ;6779   ;       clear CPU GRANT L.  During this loop, the enable RD CSR 1 L can be
                ;6780   ;       checked at the quad buffer for a low.  If this signal is high, check
                ;6781   ;       the output at the decoder which produces this signal.  If incorrect
                ;6782   ;       there, check the inputs for a low on RD CSR L and LVA 03 H, and a high
                ;6783   ;       on LVA 02 H.  If the inputs are OK, the decoder is bad.  If RD CSR 1 L
                ;6784   ;       is low, check the inputs and outputs of the buffer.  You should see
                ;6785   ;       the data pattern on the MC BUS ready for the CPU to grab on the MOV
                ;6786   ;       instruction.
                ;6787   ;
                ;6788   ;       Error 2 - Essentially the same as error 1, except for the data
                ;6789   ;       pattern.  The CSR 1 latch or quad buffer are more likely to be causing
                ;6790   ;       the failure this time.
                ;6791   ;
                ;6792   ;       Error 3 - A short between the etches between the CSR 1 latch and the
                ;6793   ;       quad buffer or internal to either chip is the most likely problem.
                ;6794   ;       Use the last paragraph of error 1 as a debug guide.
                ;6795   ;
                ;6796   ;--
                ;6797
                ;6798   T.3:
                ;6799
U 0781, B65E,15 ;6800           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;6801                                           ; STATUS WORD
U 0782, 3E80,15 ;6802           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;6803                                           ; BIT 15 INDICATES START OF TEST TO
                ;6804                                           ; CONSOLE
U 0783, 10E0,15 ;6805           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;6806   WAIT.T3.0:
U 0784, 0878,44 ;6807           JMP [WAIT.T3.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                ;6808                                           ; RESPOND
                ;6809
U 0785, 0A1A,AC ;6810           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
U 0786, 4742,15 ;6811           BIS LS[CPU] TO WR[0]            ;ADD CPU TO MODULE CODE
U 0787, 3E8C,15 ;6812           MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                ;6813                                           ; CPU AND MCT BOARDS
U 0788, B69E,15 ;6814           MOV LS[ONES] TO WR[0]           ;DISABLE ERROR CHECK ON ALL BITS
U 0789, 457A,15 ;6815           BIC LS[BIT29] TO WR[0]          ;CLEAR (CHECK) BIT 29
U 078A, C578,15 ;6816           BIC LS[BIT28] TO WR[0]          ;CLEAR (CHECK) BIT 28
U 078B, 4576,15 ;6817           BIC LS[BIT27] TO WR[0]          ;CLEAR (CHECK) BIT 27
U 078C, C574,15 ;6818           BIC LS[BIT26] TO WR[0]          ;CLEAR (CHECK) BIT 26
U 078D, C572,15 ;6819           BIC LS[BIT25] TO WR[0]          ;CLEAR (CHECK) BIT 25
U 078E, 3E8A,15 ;6820           MOV WR[0] TO LS[ERROR.MASK]     ;SET UP ERROR MASK TO CHECK BITS 29-25
                ;6821
U 078F, B69E,15 ;6822           MOV LS[ONES] TO WR[0]           ;SET UP COMPLEMENT OF EXPECTED RESULT
                ;6823   LOOP.T3.1:
U 0790, 2F82,95 ;6824           CLR WR[1]                       ;EXPECTED RESULT
U 0791, 1D45,F5 ;6825           MEM.REQ[WRITE.CSR] ADRS[CSR1] DT[LONG]
```

```
                 ;6826                                          ;EXECUTE WRITE TO CSR 1
U 0792, 0A1B,4C  ;6827          JSR [NOP.DELAY]                 ;EXECUTE 5 CYCLE DELAY TO CHECK THAT
                 ;6828                                          ; MCT IS WAITING FOR DATA REQUEST
U 0793, B29C,15  ;6829          WRITE.MEM LS[ZERO]              ;SEND OVER THE DATA PATTERN
U 0794, 9D45,75  ;6830          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                 ;6831                                          ;READ THE DATA FROM THE CSR
U 0795, 3022,15  ;6832          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0796, 0869,3C  ;6833          JSR [CHECK.RESULT]              ;AND CHECK
U 0797, 0879,04  ;6834          JMP [LOOP.T3.1]                 ;ERROR LOOP IF ENABLED
                 ;6835
U 0798, FF82,15  ;6836          INC LS[ERROR.NUMBER]            ;ERROR 2
U 0799, 2F80,15  ;6837          CLR WR[0]                       ;SET UP COMPLEMENT OF EXPECTED RESULT
                 ;6838  LOOP.T3.2:
U 079A, 369E,95  ;6839          MOV LS[ONES] TO WR[1]           ;EXPECTED RESULT
U 079B, 1D45,F5  ;6840          MEM.REQ[WRITE.CSR] ADRS[CSR1] DT[LONG]
                 ;6841                                          ;EXECUTE WRITE TO CSR 1
U 079C, 0A1B,4C  ;6842          JSR [NOP.DELAY]                 ;EXECUTE 5 CYCLE DELAY TO CHECK THAT
                 ;6843                                          ; MCT IS WAITING FOR DATA REQUEST
U 079D, 329E,15  ;6844          WRITE.MEM LS[ONES]              ;SEND OVER THE DATA PATTERN
U 079E, 9D45,75  ;6845          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                 ;6846                                          ;READ THE DATA FROM THE CSR
U 079F, 3022,15  ;6847          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 07A0, 0869,3C  ;6848          JSR [CHECK.RESULT]              ;AND CHECK
U 07A1, 0879,A4  ;6849          JMP [LOOP.T3.2]                 ;ERROR LOOP IF ENABLED
                 ;6850
U 07A2, FF82,15  ;6851          INC LS[ERROR.NUMBER]            ;ERROR 3
U 07A3, E5F8,15  ;6852          CLR LS[OS]                      ;CLEAR INDEX
                 ;6853
                 ;6854  LOOP.T3.3:
U 07A4, B6F6,95  ;6855          MOV LS[SHIFT.OS(4-0)] TO WR[1]  ;EXPECTED RESULT
U 07A5, 1D45,F5  ;6856          MEM.REQ[WRITE.CSR] ADRS[CSR1] DT[LONG]
                 ;6857                                          ;EXECUTE WRITE TO CSR 1
U 07A6, B2F6,15  ;6858          WRITE.MEM LS[SHIFT.OS(4-0)]     ;SEND OVER THE DATA PATTERN
U 07A7, 9D45,75  ;6859          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                 ;6860                                          ;READ THE DATA FROM THE CSR
U 07A8, 3022,15  ;6861          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 07A9, 0869,3C  ;6862          JSR [CHECK.RESULT]              ;AND CHECK
U 07AA, 887A,44  ;6863          JMP [LOOP.T3.3]                 ;ERROR LOOP IF ENABLED
                 ;6864
U 07AB, 7FF8,15  ;6865          INC LS[OS]                      ;INCREMENT INDEX FOR NEXT PATTERN
U 07AC, 36F9,15  ;6866          MOV LS[OS] TO WR[2]             ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                 ;6867          BIT LS[BIT5] WITH WR[2],        ;SEE IF OS HAS INCREMENTED TO 20(H)
U 07AD, D94B,35  ;6868           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 07AE, 087A,49  ;6869          JMP.IF[BITS.CLR] TO [LOOP.T3.3] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                 ;6870
                 ;6871  END.T3:
```

```
;6872    .PAGE   "  TEST 4 Branch on LVA00, ECC DIS, and DIAG CHECK Test (M8391  MCT Module)"
;6873    ;++
;6874    ;
;6875    ; Test Description:
;6876    ;
;6877    ;       This test checks the branch logic used to control u-address bits 0 and
;6878    ;       1 (BEN 0 and BEN 1 respectively) for three inputs:  BEN 0 branch on
;6879    ;       LVA00 and DIAG CHECK, and BEN 1 branch on ECC DIS.  DIAG CHECK and ECC
;6880    ;       DIS come from CSR 1 and are directly writable via the WRITE.CSR memory
;6881    ;       request instruction.  LVA00 comes directly from the VAR.  The data
;6882    ;       path is as follows:  The CPU issues a MEM.REQ with MF=RD.MAINT.BRANCH.
;6883    ;       CK (1BH).  The dispatch in the memory controller proceeds as described
;6884    ;       in the beginning of this listing.  After the dispatch the following
;6885    ;       sequence occurs:  The first u-word sets memory busy and branches on
;6886    ;       LVA00 (BEN 0, u-address bit 0).  If LVA00 is clear (0) then the memory
;6887    ;       u-code will keep memory busy set for one more cycle, then return the
;6888    ;       VAR contents on the MC BUS (using the end of the READ.MAINT.ADRS rou-
;6889    ;       tine).  The CPU will then check that the VAR contents are unchanged.
;6890    ;       If LVA00 is set (1) then memory busy is kept set and the VAR is incr-
;6891    ;       mented.  Also in this u-word, a branch is made according to the values
;6892    ;       of DIAG.CHK and ECC.DIS.  DIAG.CHK and ECC.DIS were already set up via
;6893    ;       the write CSR routine prior to entering this test.  The next u-words
;6894    ;       increment the VAR according to the following description and return
;6895    ;       the value of the VAR to the CPU for checking:  DIAG.CHK set, ECC.DIS
;6896    ;       set - no inc; DIAG.CHK set, ECC.DIS clear - 1 inc; DIAG.CHK clear,
;6897    ;       ECC.DIS set - 2 incs; DIAG.CHK clear, ECC.DIS clear - 3 incs.  The CPU
;6898    ;       reads the value of the VAR to determine if the branch occured
;6899    ;       correctly.
;6900    ;
;6901    ; Assumptions:
;6902    ;
;6903    ;       It is assumed that all previous tests have run successfully.
;6904    ;
;6905    ; Test Steps:
;6906    ;
;6907    ;       1)  Set up error number, and module number in LS (for error printouts)
;6908    ;           and clear previous error number in LS.
;6909    ;       2)  Set up error mask to check low byte only, and clear WR 1.
;6910    ;       3)  Execute MEM.REQ with DT=longword and MF=RD.MAINT.BRANCH.CHK using
;6911    ;           the location in LS containing 0's as data.
;6912    ;       4)  Execute MOV MEM.DATA TO WR[0] and check VAR for 0.
;6913    ;       5)  Write CSR 1 to set DIAG.CHK and ECC.DIS.  Write WR 1 with a 2(H).
;6914    ;       6)  Execute MEM.REQ with DT=longword and MF=RD.MAINT.BRANCH.CHK using
;6915    ;           the location in LS containing a 1 (bit 0 set) as data.
;6916    ;       7)  Execute MOV MEM.DATA TO WR[0] and check VAR for a 2(H).
;6917    ;       8)  Write CSR 1 to clear DIAG.CHK and set ECC.DIS.  Write WR 1 with a
;6918    ;           4(H) (expected data).
;6919    ;       9)  Repeat steps 6 and 7 checking for a 4(H) in the VAR.
;6920    ;       10) Write CSR 1 to set DIAG.CHK and clear ECC.DIS.  Write WR 1 with a
;6921    ;           6(H) (expected data).
;6922    ;       11) Repeat steps 6 and 7 checking for an 6(H) in the VAR.
;6923    ;       12) Write CSR 1 to clear DIAG.CHK and ECC.DIS.  Write WR 1 with an 8(H)
;6924    ;           (expected data).
;6925    ;       13) Repeat steps 6 and 7 checking for an 8(H) in the VAR.
;6926    ;
```

```
;6927    ; Errors:
;6928    ;
;6929    ;       NOTE:  The expected and received data printed in the error report is
;6930    ;       the expected and received value of the VAR.  This value is useful in
;6931    ;       determining where the memory microcode branched.  See debug section.
;6932    ;
;6933    ;       Error 1 - Branch code BEN0=LVA00 failed with LVA00=0.
;6934    ;       Error 2 - Branch code BEN0=LVA00 or BEN0=LDIAG.CHK and BEN1=LECC.DIS
;6935    ;                 failed with LVA00=1, DIAG.CHK=1, and ECC.DIS=1.
;6936    ;       Error 3 - Branch code BEN0=LDIAG.CHK and BEN1=LECC.DIS failed with
;6937    ;                 DIAG.CHK=0 and ECC.DIS=1.
;6938    ;       Error 4 - Branch code BEN0=LDIAG.CHK and BEN1=LECC.DIS failed with
;6939    ;                 DIAG.CHK=1 and ECC.DIS=0.
;6940    ;       Error 5 - Branch code BEN0=LDIAG.CHK and BEN1=LECC.DIS failed with
;6941    ;                 DIAG.CHK=0 and ECC.DIS=0.
;6942    ;
;6943    ; Debug:
;6944    ;
;6945    ;       Two methods of debug will be explained.  If the MCT module is in a
;6946    ;       test station then the MCT can be single stepped making debug easier
;6947    ;       in some cases.  Otherwise only the CPU can be single stepped and this
;6948    ;       method will be explained also.  When single stepping the memory con-
;6949    ;       troller will help in debug, this section will explicitly suggest it.
;6950    ;
;6951    ;       Error 1 - A failure here indicates and problem with the BEN0 8 to 1
;6952    ;       mux or its inputs.  Check the input signal LVA 00 H for a low during
;6953    ;       the MEM.REQ instruction.  This signal has already been tested at its
;6954    ;       source so an open etch, indicated by a float, is possible.  If this
;6955    ;       signal is OK, check the C 09, C 10, and C 11 inputs.  If MCT single
;6956    ;       step is available, step to the first u-word after the dispatch where
;6957    ;       BEN 0=LVA00.  The select signals C 11 H, C 10 H, and C 09 H should
;6958    ;       equal 001(B) respectively.  If these inputs are correct, the MUX is
;6959    ;       probably bad.  If MCT single step is not available, the inputs will
;6960    ;       have to be checked while looping on error.
;6961    ;
;6962    ;       Error 2 - This error indicates a problem with either the BEN0 8 to 1
;6963    ;       mux, the BEN1 8 to 1 mux, or their inputs.  If the received data (VAR)
;6964    ;       is 0, then BEN0=LVA00 failed.  Single step the CPU to the MEM.REQ
;6965    ;       instruction that sends data of 1 to the VAR and check the signal LVA
;6966    ;       00 H at the BEN 0 mux for a high.  If this is OK the select lines
;6967    ;       should be checked.  If MCT single step is available, step to the
;6968    ;       memory U-word after the dispatch (BEN0=LVA00) and check C 11 H, C 10
;6969    ;       H, and C 09 H for a 001(B) respectively.  If the inputs are OK, the
;6970    ;       MUX is probably bad.  If received data is greater than 2, then
;6971    ;       BEN0=LDIAG.CHK or BEN1=LECC.DIS failed.  If received data=4(H), then
;6972    ;       BEN1=LECC.DIS failed.  If received data=6(H), then BEN0=LECC.DIS
;6973    ;       failed.  If received data=8(H), then both failed.  Check the inputs
;6974    ;       to the appropriate mux as described before.  If MCT single step is
;6975    ;       available, step to the second memory u-word after the dispatch address
;6976    ;       to check the inputs.  Otherwise loop on error will be necessary.  The
;6977    ;       inputs to the BEN 0 mux should be 100(B) on the select and L DIAG CHK
;6978    ;       H should be high. The inputs to the BEN 1 mux should be 011(B) on the
;6979    ;       select and L ECC DIS H should be high.  Suspect the mux if the inputs
;6980    ;       are OK.    Note:  The signals L DIAG CHK H and L ECC DIS H can be
;6981    ;       checked without MCT single step by single stepping the CPU to the
```

```
                ;6982   ;       MEM.REQ that calls this test.  These signals will be valid after the
                ;6983   ;       write to CSR 1.
                ;6984   ;
                ;6985   ;       Error 3 - If received data=2(H) then L DIAG CHK H probably failed to
                ;6986   ;       go low at the BEN 0 mux.  Any other received data value probably
                ;6987   ;       indicates A problem with the select input to one of the two muxes or a
                ;6988   ;       failure in the mux itself.  If MCT single step is available, step the
                ;6989   ;       memory u-code to the second u-word after the dispatch address
                ;6990   ;       (BEN0=LDIAG.CHK, BEN1=LECC.DIS) and check the inputs to the mux for
                ;6991   ;       the following:  BEN 0 mux select should be 100(B) and L DIAG CHK H
                ;6992   ;       should be low.  BEN 1 mux select should be 011(B) and L ECC DIS H
                ;6993   ;       should be high.  If MCT single step is not available, then loop on
                ;6994   ;       error must be used to check select inputs.
                ;6995   ;         Note:  The signals L DIAG CHK H and L ECC DIS H can be checked with-
                ;6996   ;       out MCT single step by single stepping the CPU to the MEM.REQ that
                ;6997   ;       calls this test.  These signals will be valid after the write to CSR
                ;6998   ;       1.
                ;6999   ;
                ;7000   ;       Error 4 - If received data=2(H) then L ECC DIS H probably failed to go
                ;7001   ;       low at the BEN 1 mux.  Any other received data value probably
                ;7002   ;       indicates A problem with the select input to one of the two muxes or a
                ;7003   ;       failure in the mux itself.  If MCT single step is available, step the
                ;7004   ;       memory u-code to the second u-word after the dispatch address
                ;7005   ;       (BEN0=LDIAG.CHK, BEN1=LECC.DIS) and check the inputs to the mux for
                ;7006   ;       the following:  BEN 0 mux select should be 100(B) and L DIAG CHK H
                ;7007   ;       should be high.  BEN 1 mux select should be 011(B) and L ECC DIS H
                ;7008   ;       should be low.  If MCT single step is not available, then loop on
                ;7009   ;       error must be used to check select inputs.
                ;7010   ;         Note:  The signals L DIAG CHK H and L ECC DIS H can be checked with-
                ;7011   ;       out MCT single step by single stepping the CPU to the MEM.REQ that
                ;7012   ;       calls this test.  These signals will be valid after the write to CSR
                ;7013   ;       1.
                ;7014   ;
                ;7015   ;       Error 5 - If this is the first error, most likely a mux is failing.
                ;7016   ;       If received data=4(H), then suspect the BEN 1 mux.  If received data=
                ;7017   ;       6(H), then suspect the BEN 0 mux.  If received data=2(H), then
                ;7018   ;       suspect both muxes (this is very unlikely).
                ;7019   ;
                ;7020   ;--
                ;7021
                ;7022   T.4:
                ;7023
U 07AF, B65E,15 ;7024           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;7025                                           ; STATUS WORD
U 07B0, 3E80,15 ;7026           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;7027                                           ; BIT 15 INDICATES START OF TEST TO
                ;7028                                           ; CONSOLE
U 07B1, 10E0,15 ;7029           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;7030   WAIT.T4.0:
U 07B2, 087B,24 ;7031           JMP [WAIT.T4.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                ;7032                                           ; RESPOND
                ;7033
U 07B3, 0A1A,AC ;7034           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
U 07B4, B62C,15 ;7035           MOV LS[#FFFFFF00] TO WR[0]      ;SET UP DATA FOR ERROR MASK
U 07B5, 3E8A,15 ;7036           MOV WR[0] TO LS[ERROR.MASK]     ;SET UP ERROR MASK TO CHECK LOW BYTE
```

```
                ;7037
                ;7038   LOOP.T4.1:
U 07B6, 2F82,95 ;7039           CLR WR[1]                           ;EXPECTED DATA
U 07B7, 1E9D,F5 ;7040           MEM.REQ[READ.MAINT.BR.CHECK] ADRS[ZERO] DT[LONG]
                ;7041                                               ;EXECUTE BRANCH TEST WITH LVA00=0
U 07B8, 3022,15 ;7042           MOV MEM.DATA TO WR[0]               ;GET RESULT (VAR)
U 07B9, 0869,3C ;7043           JSR [CHECK.RESULT]                  ;CHECK FOR ERROR
U 07BA, 887B,64 ;7044           JMP [LOOP.T4.1]                     ;ERROR LOOP IF ENABLED
                ;7045
U 07BB, FF82,15 ;7046           INC LS[ERROR.NUMBER]                ;ERROR 2
U 07BC, 3672,15 ;7047           MOV LS[ECC.DIS] TO WR[0]            ;GOING TO SET ECC DISABLE IN CSR1
U 07BD, 4774,15 ;7048           BIS LS[DIAG.CHK] TO WR[0]           ;SET DIAG CHK BIT ALSO
U 07BE, 8A17,FC ;7049           JSR [WRITE.CSR1]                    ;WRITE CSR1 WITH DATA FROM WR 0
                ;7050   LOOP.T4.2:
U 07BF, B642,95 ;7051           MOV LS[#2] TO WR[1]                 ;EXPECTED DATA
U 07C0, 9E41,F5 ;7052           MEM.REQ[READ.MAINT.BR.CHECK] ADRS[#1] DT[LONG]
                ;7053                                               ;EXECUTE BRANCH TEST WITH LVA00=1,
                ;7054                                               ; ECC DIS=1 AND DIAG CHK=1
U 07C1, 3022,15 ;7055           MOV MEM.DATA TO WR[0]               ;GET RESULT (VAR)
U 07C2, 0869,3C ;7056           JSR [CHECK.RESULT]                  ;CHECK FOR ERROR
U 07C3, 887B,F4 ;7057           JMP [LOOP.T4.2]                     ;ERROR LOOP IF ENABLED
                ;7058
U 07C4, FF82,15 ;7059           INC LS[ERROR.NUMBER]                ;ERROR 3
U 07C5, 3672,15 ;7060           MOV LS[ECC.DIS] TO WR[0]            ;GOING TO SET ECC DISABLE IN CSR
U 07C6, 8A17,FC ;7061           JSR [WRITE.CSR1]                    ;WRITE CSR1 WITH DATA FROM WR 0
                ;7062   LOOP.T4.3:
U 07C7, B644,95 ;7063           MOV LS[#4] TO WR[1]                 ;EXPECTED DATA
U 07C8, 9E41,F5 ;7064           MEM.REQ[READ.MAINT.BR.CHECK] ADRS[#1] DT[LONG]
                ;7065                                               ;EXECUTE BRANCH TEST WITH LVA00=1,
                ;7066                                               ; ECC DIS=1 AND DIAG CHK=0
U 07C9, 3022,15 ;7067           MOV MEM.DATA TO WR[0]               ;GET RESULT (VAR)
U 07CA, 0869,3C ;7068           JSR [CHECK.RESULT]                  ;CHECK FOR ERROR
U 07CB, 887C,74 ;7069           JMP [LOOP.T4.3]                     ;ERROR LOOP IF ENABLED
                ;7070
U 07CC, FF82,15 ;7071           INC LS[ERROR.NUMBER]                ;ERROR 4
U 07CD, 3674,15 ;7072           MOV LS[DIAG.CHK] TO WR[0]           ;GOING TO SET DIAG CHK BIT IN CSR
U 07CE, 8A17,FC ;7073           JSR [WRITE.CSR1]                    ;WRITE CSR1 WITH DATA FROM WR 0
                ;7074   LOOP.T4.4:
U 07CF, B644,95 ;7075           MOV LS[#4] TO WR[1]                 ;EXPECTED DATA IS 6.  SET BIT 2
U 07D0, C742,95 ;7076           BIS LS[BIT1] TO WR[1]               ;NOW EXPECTED DATA=6
U 07D1, 9E41,F5 ;7077           MEM.REQ[READ.MAINT.BR.CHECK] ADRS[#1] DT[LONG]
                ;7078                                               ;EXECUTE BRANCH TEST WITH LVA00=1,
                ;7079                                               ; ECC DIS=0 AND DIAG CHK=1
U 07D2, 3022,15 ;7080           MOV MEM.DATA TO WR[0]               ;GET RESULT (VAR)
U 07D3, 0869,3C ;7081           JSR [CHECK.RESULT]                  ;CHECK FOR ERROR
U 07D4, 087C,F4 ;7082           JMP [LOOP.T4.4]                     ;ERROR LOOP IF ENABLED
                ;7083
U 07D5, FF82,15 ;7084           INC LS[ERROR.NUMBER]                ;ERROR 5
U 07D6, 0A17,EC ;7085           JSR [CLEAR.CSR1]                    ;WRITE CSR1 WITH DATA OF 0
                ;7086   LOOP.T4.5:
U 07D7, 3646,95 ;7087           MOV LS[#8] TO WR[1]                 ;EXPECTED DATA
U 07D8, 9E41,F5 ;7088           MEM.REQ[READ.MAINT.BR.CHECK] ADRS[#1] DT[LONG]
                ;7089                                               ;EXECUTE BRANCH TEST WITH LVA00=1,
                ;7090                                               ; ECC DIS=0 AND DIAG CHK=0
U 07D9, 3022,15 ;7091           MOV MEM.DATA TO WR[0]               ;GET RESULT (VAR)
```

```
U 07DA, 0869,3C ;7092            JSR [CHECK.RESULT]              ;CHECK FOR ERROR
U 07DB, 087D,74 ;7093            JMP [LOOP.T4.5]                 ;ERROR LOOP IF ENABLED
                ;7094   END.T4:
```

```
;7095    .PAGE   " UNIBUS address, data and control logic tests"
;7096
;7097    .toc    "  TEST 5 UNIBUS Address Test (M8391  MCT Module)"
;7098    ;++
;7099    ;
;7100    ; Test Description:
;7101    ;
;7102    ;       This test checks the UBS AXX inputs to the VAR PALs and the control
;7103    ;       logic and VAR PALs themselves.  The VAR PALs will be totally tested
;7104    ;       after this test has completed.  The data path is as follows:  The CPU
;7105    ;       issues a MEM.REQ with the MF field indicating RD.MAINT.UBS (1F).  The
;7106    ;       dispatch occurs as described in the beginning of this listing.  At the
;7107    ;       dispatch address memory busy is set, IB BSY is set, address physical
;7108    ;       is enabled, UB ADR, UB data, and DATIP is enabled, and a branch on
;7109    ;       DIAG CHK and ECC DIS is executed.  Prior to executing the MEM.REQ
;7110    ;       instruction, CSR 1 was written to clear both DIAG CHK and ECC DIS.  So
;7111    ;       the branch in memory u-code at this point will go to the code that
;7112    ;       tests the UB ADR logic.  The first memory u-word after the branch
;7113    ;       keeps memory busy and IB BSY asserted, and keeps UB ADR enabled.  UB
;7114    ;       ADR EN L has put the contents of the VAR bits 00 through 08 and PA
;7115    ;       bits 10  through 17 on the UB ADR A00 through UB ADR A17 lines so they
;7116    ;       are now present at the input to the VAR PAL's UB ADR AXX inputs (the
;7117    ;       VAR was loaded by data sent with the MEM.REQ CPU u-instruction).  The
;7118    ;       next u-cycle enables the VAR PAL to load the UB ADR inputs into its
;7119    ;       internal latches, and keeps UB ADR EN, memory busy, and IB BSY
;7120    ;       enabled.  The next cycle reads the data back to the CPU by placing the
;7121    ;       VAR contents on the MC BUS as in the READ.MAINT.ADR test.  Data
;7122    ;       patterns used are all 1's, all 0's and shifting 1's.  VAR bits 18 and
;7123    ;       31 are forced low by hardware and bits 19-30 are forced high.  These
;7124    ;       bits are also checked (they come back as bits 18-29 high, 30 and 17
;7125    ;       low.)
;7126    ;
;7127    ; Assumptions:
;7128    ;
;7129    ;       It is assumed that the previous VAR tests have run successfully.
;7130    ;
;7131    ; Test Steps:
;7132    ;
;7133    ;       1)  Set up error mask, error number, and module number in LS (for
;7134    ;           error printouts) and clear previous error number in LS.
;7135    ;       2)  Execute a write to CSR 1 to clear ECC DIS and DIAG CHK.  This data
;7136    ;           is used by the memory u-code for branching to the correct test.
;7137    ;       3)  Load WR 3 with data pattern from LS.
;7138    ;       4)  BIC bits 31 and 18, BIS bits 19-30 (these bits are always forced
;7139    ;           this way by the hardware before the right shift).  Do a ROR WR 3
;7140    ;           to get expected data and put in WR 1.
;7141    ;       5)  Issue a MEM.REQ with DT=longword and MF=READ.MAINT.UBS using the
;7142    ;           data from the LS location containing the data pattern as the test
;7143    ;           pattern.
;7144    ;       6)  Execute MOV MEM to WR[0] to read result and check.  Bits 17-00 are
;7145    ;           UB DATA (this comes back shifted right 1 bit, so the result is on
;7146    ;           bits 16-00 and 31).  The other bits are checked for a 1 on VAR
;7147    ;           bits 19-30 and a 0 on bits 31 and 18 (before shift).
;7148    ;       7)  Repeat steps 3 through 5 with each data pattern.
;7149    ;
```

```
                ;7150   ; Errors:
                ;7151   ;
                ;7152   ;       Error 1 - UNIBUS Address test failed with data of all 1's.
                ;7153   ;       Error 2 - UNIBUS Address test failed with data of all 0's.
                ;7154   ;       Error 3 - UNIBUS Address test failed with data of shifting 1's.
                ;7155   ;
                ;7156   ; Debug:
                ;7157   ;
                ;7158   ;       Two methods of debug will be explained.  If the MCT module is in a
                ;7159   ;       test station then the MCT can be single stepped making debug easier
                ;7160   ;       in some cases.  Otherwise only the CPU can be single stepped and this
                ;7161   ;       method will be explained also.  When single stepping the memory con-
                ;7162   ;       troller will help in debug, this section will explicitly suggest it.
                ;7163   ;
                ;7164   ;       Error 1 - This error indicates a problem with the VAR PALS or the UB
                ;7165   ;       ADR logic.  If MCT single step is available, step to the memory
                ;7166   ;       u-cycle which enables the VAR load of UB data and check the inputs to
                ;7167   ;       the VAR PAL for a high on VAR MUX SEL H and VAR LOAD H.  Check the UB
                ;7168   ;       ADR AXX H inputs for a high.  If the UB ADR inputs are incorrect,
                ;7169   ;       check the UBS ADR drivers for a high on the AND gate enables (UB ADR
                ;7170   ;       EN L provides this) and a high on the LVA XX H inputs to the other leg
                ;7171   ;       of the AND gate.  Also check the inputs to the inverter before UBS ADR
                ;7172   ;       AXX H is produced.  If the inputs to the VAR PALs are OK, then suspect
                ;7173   ;       the PAL itself.  If the failing bits are in the range 20-31 (after the
                ;7174   ;       shift) the most likely failure is the VAR PAL itself which produces
                ;7175   ;       LVA 19 H through LVA 30 H.  Check the hardwire grounds on pin 3 of hte
                ;7176   ;       VAR PAL which produces LVA 31 H and pin 9 of the VAR PAL which
                ;7177   ;       produces LVA 18 H if bits 0 or 19 of the result are failing.  If MCT
                ;7178   ;       single step is not available, thenthese inputs will have to be scoped
                ;7179   ;       while looping on error.
                ;7180   ;
                ;7181   ;       Error 2 - Same as error 1 except input to VAR PAL's should be all 0's
                ;7182   ;       on UB ADR AXX.
                ;7183   ;         Another possibility is that the signal ADDR PH L is going away after
                ;7184   ;       it is set.  This would show up as an error in received data bits 10
                ;7185   ;       through 18 being high instead of low.  If MCT single step is
                ;7186   ;       available, step to the memory u-cycle that loads the VAR PAL with UB
                ;7187   ;       ADR and check ADR PH L for a low.  If this is incorrect, check the
                ;7188   ;       inputs to the MISC CONTROL PAL for CONT FUNC LAT L low and TB DATA EN
                ;7189   ;       L high.  If these are correct, suspect the MISC CONTROL PAL itself.
                ;7190   ;
                ;7191   ;       Error 3 - See error 2.  Difference is in data pattern, which is a
                ;7192   ;       shifting 1 pattern.  Most likely cause is an etch short.
                ;7193   ;
                ;7194   ;--
                ;7195
                ;7196   T.5:
                ;7197
U 07DC, B65E,15 ;7198           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;7199                                           ; STATUS WORD
U 07DD, 3E80,15 ;7200         . MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;7201                                           ; BIT 15 INDICATES START OF TEST TO
                ;7202                                           ; CONSOLE
U 07DE, 10E0,15 ;7203           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;7204   WAIT.T5.0:
```

```
U 07DF, 887D,F4 ;7205          JMP [WAIT.T5.0]                  ;LOOP HERE WAITING FOR CONSOLE TO
                ;7206                                           ; RESPOND
                ;7207
U 07E0, 0A1A,9C ;7208          JSR [SETUP]                      ;SET UP MASKS, MODULE CODE ETC.
                ;7209
U 07E1, 5F7F,95 ;7210          MCOM LS[BIT31] TO WR[3]          ;CLEAR BIT 31, SET OTHERS IN WR 3
U 07E2, 4565,95 ;7211          BIC LS[BIT18] TO WR[3]           ;CLEAR BIT 18 (THE MCT HARDWARE CLEARS
                ;7212                                           ; THESE BITS, SO EXPECTED MUST BE CLEAR)
U 07E3, 2341,95 ;7213          ROR WR[3]                        ;CORRECT FOR HARDWARE SHIFT
                ;7214  LOOP.T5.1:
U 07E4, 2006,95 ;7215          MOV WR[3] TO WR[1]               ;SET UP EXPECTED DATA
U 07E5, 1F9F,F5 ;7216          MEM.REQ[READ.MAINT.UBS] ADRS[ONES] DT[LONG]
                ;7217                                           ;WRITE VAR TO UNIBUS ADDRESS LINES (DATA
                ;7218                                           ; OF ONES)
U 07E6, 3022,15 ;7219          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 07E7, 0869,3C ;7220          JSR [CHECK.RESULT]               ;AND CHECK FOR ERRORS
U 07E8, 087E,44 ;7221          JMP [LOOP.T5.1]                  ;ERROR LOOP IF ENABLED
                ;7222
U 07E9, FF82,15 ;7223          INC LS[ERROR.NUMBER]             ;ERROR 2
U 07EA, B635,95 ;7224          MOV LS[UBS.SET] TO WR[3]         ;SET BITS IN WR 3 THAT ARE SET BY MCT
                ;7225                                           ; HARDWARE
U 07EB, 2341,95 ;7226          ROR WR[3]                        ;CORRECT FOR HARDWARE SHIFT
                ;7227  LOOP.T5.2:
U 07EC, 2006,95 ;7228          MOV WR[3] TO WR[1]               ;SET UP EXPECTED DATA
U 07ED, 9F9D,F5 ;7229          MEM.REQ[READ.MAINT.UBS] ADRS[ZERO] DT[LONG]
                ;7230                                           ;WRITE VAR TO UNIBUS ADDRESS LINES (DATA
                ;7231                                           ; OF ZERO)
U 07EE, 3022,15 ;7232          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 07EF, 0869,3C ;7233          JSR [CHECK.RESULT]               ;AND CHECK FOR ERRORS
U 07F0, 887E,C4 ;7234          JMP [LOOP.T5.2]                  ;ERROR LOOP IF ENABLED
                ;7235
U 07F1, FF82,15 ;7236          INC LS[ERROR.NUMBER]             ;ERROR 3
U 07F2, E5F8,15 ;7237          CLR LS[OS]                       ;CLEAR INDEX
                ;7238  REPEAT.T5.3:
U 07F3, 36F7,95 ;7239          MOV LS[SHIFT.OS(4-0)] TO WR[3]   ;LOAD WR 3 WITH DATA PATTERN
U 07F4, C735,95 ;7240          BIS LS[UBS.SET] TO WR[3]         ;SET BITS THAT ARE SET BY MCT HARDWARE
U 07F5, C57F,95 ;7241          BIC LS[BIT31] TO WR[3]           ;CLEAR BITS THAT ARE CLEARED BY MCT
                ;7242                                           ; HARDWARE
U 07F6, 4565,95 ;7243          BIC LS[BIT18] TO WR[3]           ;CLEAR BITS THAT ARE CLEARED BY MCT
                ;7244                                           ; HARDWARE
U 07F7, 2341,95 ;7245          ROR WR[3]                        ;CORRECT FOR HARDWARE SHIFT
                ;7246  LOOP.T5.3:
U 07F8, 2006,95 ;7247          MOV WR[3] TO WR[1]               ;SET UP EXPECTED DATA
U 07F9, 9FF7,F5 ;7248          MEM.REQ[READ.MAINT.UBS] ADRS[SHIFT.OS(4-0)] DT[LONG]
                ;7249                                           ;WRITE VAR TO UNIBUS ADDRESS LINES (DATA
                ;7250                                           ; OF SHIFTING ONES)
U 07FA, 3022,15 ;7251          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 07FB, 0869,3C ;7252          JSR [CHECK.RESULT]               ;AND CHECK FOR ERRORS
U 07FC, 887F,84 ;7253          JMP [LOOP.T5.3]                  ;ERROR LOOP IF ENABLED
                ;7254
U 07FD, 7FF8,15 ;7255          INC LS[OS]                       ;INCREMENT INDEX FOR NEXT PATTERN
U 07FE, 36F9,15 ;7256          MOV LS[OS] TO WR[2]              ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                ;7257          BIT LS[BIT5] WITH WR[2],         ;SEE IF OS HAS INCREMENTED TO 20(H)
U 07FF, D94B,35 ;7258           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 0800, 887F,39 ;7259          JMP.IF[BITS.CLR] TO [REPEAT.T5.3]  ;REPEAT WITH NEXT PATTERN IF NOT DONE
```

```
                ;7260   T5.END:
```

```
;7261   .PAGE   "  TEST 6 UNIBUS Data Test (M8391  MCT Module and M8394 WCS Module)"
;7262   ;++
;7263   ;
;7264   ; Test Description:
;7265   ;
;7266   ;       This test checks the UNIBUS data logic using the turnaround logic on
;7267   ;       the WCS module.  All control signals and the test pattern data come
;7268   ;       from the MCT module.  The data path is as follows:  A MEM.REQ CPU
;7269   ;       u-instruction is issued with DT=longword and MF=READ.MAINT.UBS (1FH).
;7270   ;       The dispatch occurs as described in the beginning of this listing.
;7271   ;       The memory u-cycle at the dispatch address sets memory busy and IB BSY,
;7272   ;       and enables address physical, DATIP, UB address, and UB data.  It then
;7273   ;       branches according to DIAG CHK and ECC DIS, which was set up prior to
;7274   ;       the MEM.REQ.  For this test, DIAG CHK is cleared and ECC DIS is set.
;7275   ;       At the branch destination, the UB DIR latches on the WCS board are
;7276   ;       opened, the UB DATA transceivers on the WCS module are enabled to
;7277   ;       receive a test pattern from the MC BUS, the MC BUS is enabled with
;7278   ;       data from the VAR, and memory busy and IB BSY are kept asserted.  The
;7279   ;       following cycle repeats the previous commands to allow data to
;7280   ;       propagate and settle.  The next memory u-cycle closes the UB latch,
;7281   ;       and leaves UB data, memory busy, and IB BSY enabled.  The following
;7282   ;       cycle enables the UB latch back onto the MC BUS, and keeps memory busy
;7283   ;       and IB BSY asserted.  The next memory u-cycle loops waiting for CPU
;7284   ;       grant to become deasserted while keeping the UB data latch enabled on
;7285   ;       the MC BUS.  CPU grant goes away when the CPU executes its MOV
;7286   ;       MEM.DATA TO WR[0].  The final memory u-cycle checks for another CPU
;7287   ;       request and goes to the idle loop.  Data patterns used are all 1's,
;7288   ;       all 0's, and shifting 1's.  Only 16 bits will be checked (received
;7289   ;       data bits 14-00 and 31).
;7290   ;
;7291   ; Assumptions:
;7292   ;
;7293   ;       It is assumed that previus tests have run successfully.  Note that
;7294   ;       most of the logic that this test checks is on the WCS module, but that
;7295   ;       all control signals and data patterns originate on the MCT module.
;7296   ;
;7297   ; Test Steps:
;7298   ;
;7299   ;       1)  Set up error number, and module number in LS (for error printouts)
;7300   ;           and clear previous error number in LS.
;7301   ;       2)  Set up error mask to check 16 bits, write CSR 1 to set the ECC DIS
;7302   ;           bit and clear the DIAG CHK bit (these bits will be used to branch
;7303   ;           to the correct test).
;7304   ;       3)  Load WR 1 with a data pattern from LS.
;7305   ;       4)  Shift WR[1] right 1 bit.
;7306   ;       5)  Issue MEM.REQ with DT=longword and MF=READ.MAINT.UBS using the
;7307   ;           location in LS containing the data pattern as the source of data.
;7308   ;       6)  Execute MOV MEM.DATA TO WR[0] to read result and check.
;7309   ;       7)  Repeat steps 3 through 6 with other data patterns.
;7310   ;
;7311   ; Errors:
;7312   ;
;7313   ;       Error 1 - UNIBUS data logic failed with data of all 1's.
;7314   ;       Error 2 - UNIBUS data logic failed with data of all 0's.
;7315   ;       Error 3 - UNIBUS data logic failed with data of shifting 1's.
```

```
                    ;7316   ;
                    ;7317   ; Debug:
                    ;7318   ;
                    ;7319   ;       Two methods of debug will be explained.  If the MCT module is in a
                    ;7320   ;       test station then the MCT can be single stepped making debug easier
                    ;7321   ;       in some cases.  Otherwise only the CPU can be single stepped and this
                    ;7322   ;       method will be explained also.  When single stepping the memory con-
                    ;7323   ;       troller will help in debug, this section will explicitly suggest it.
                    ;7324   ;
                    ;7325   ;       Error 1 - This error indicates a problem with the UNIBUS data logic or
                    ;7326   ;       control signals.  Most of the logic tested here is on the WCS module.
                    ;7327   ;       If MCT single step is available, step to the first u-cycle after the
                    ;7328   ;       dispatch address (the first of the two cycles that do the same thing).
                    ;7329   ;       On the WCS module check that the signals UB DATA EN L is low and UB
                    ;7330   ;       DIR LATCH L is high.  If either of these signals is incorrect, trace
                    ;7331   ;       them back to the control store proms on the MCT module.  If these
                    ;7332   ;       signals are OK, check the D inputs to the 8 bit latches for all highs.
                    ;7333   ;       If incorrect, trace back to the MC BUS.  If these inputs are OK, the
                    ;7334   ;       problem probably is the 8 bit latch itself or the BUS MC connections to
                    ;7335   ;       the module fingers.  The output of the 8 bit latches can be checked by
                    ;7336   ;       allowing the MCT to run freely (no single step) and single stepping the
                    ;7337   ;       CPU to the MEM.REQ instruction.  The signal UB DIR EN L should now be
                    ;7338   ;       low and UB DIR LATCH L should also be low.  These two signals can be
                    ;7339   ;       traced back to the MCT control store PROMS.
                    ;7340   ;
                    ;7341   ;       Error 2 - The same procedure can be followed as in error 1.  The inputs
                    ;7342   ;       to the 8 bit latches should now be low.  If this is the first error, a
                    ;7343   ;       likely possibility is that UB DIR EN L is failing to go low.  Single
                    ;7344   ;       stepping the CPU to the MEM.REQ instruction will allow testing this.
                    ;7345   ;       If this signal is OK, then trace the error as in error 1.
                    ;7346   ;         One other possibility is that a defective device on the UNIBUS is
                    ;7347   ;       holding the bus low (asserted) so that a 1 is seen on the output.
                    ;7348   ;
                    ;7349   ;       Error 3 - Suspect an etch short between bits or a bad 8 bit latch on
                    ;7350   ;       the MCT module.
                    ;7351   ;
                    ;7352   ;--
                    ;7353
                    ;7354   T.6:
                    ;7355
U 0801, B65E,15     ;7356           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;7357                                           ; STATUS WORD
U 0802, 3E80,15     ;7358           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;7359                                           ; BIT 15 INDICATES START OF TEST TO
                    ;7360                                           ; CONSOLE
U 0803, 10E0,15     ;7361           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;7362   WAIT.T6.0:
U 0804, 8880,44     ;7363           JMP [WAIT.T6.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                    ;7364                                           ; RESPOND
                    ;7365
U 0805, 0A1A,AC     ;7366           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
U 0806, C740,15     ;7367           BIS LS[WCS] TO WR[0]            ;ADD WCS BOARD TO MODULE PRINTOUT
U 0807, 3E8C,15     ;7368           MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                    ;7369                                           ; MCT BOARD AND WCS BOARD
U 0808, 3636,15     ;7370           MOV LS[UBS.DATA] TO WR[0]       ;SET UP DATA FOR ERROR MASK
```

```
U 0809, 3E8A,15 ;7371          MOV WR[0] TO LS[ERROR.MASK]     ;STORE ERROR MASK TO CHECK BITS 14-00
                ;7372                                          ; AND BIT 31
                ;7373
U 080A, 3672,15 ;7374          MOV LS[ECC.DIS] TO WR[0]        ;SET UP TO SET ECC DISABLE BIT IN CSR
U 080B, 8A17,FC ;7375          JSR [WRITE.CSR1]                ;WRITE CSR 1 WITH DATA IN WR 0
                ;7376
                ;7377  LOOP.T6.1:
U 080C, 369E,95 ;7378          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 080D, 1F9F,F5 ;7379          MEM.REQ[READ.MAINT.UBS] ADRS[ONES] DT[LONG]
                ;7380                                          ;WRITE UNIBUS DATA OF ONES
U 080E, 3022,15 ;7381          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 080F, 0869,3C ;7382          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0810, 0880,C4 ;7383          JMP [LOOP.T6.1]                 ;ERROR LOOP IF ENABLED
                ;7384
U 0811, FF82,15 ;7385          INC LS[ERROR.NUMBER]            ;ERROR 2
                ;7386  LOOP.T6.2:
U 0812, 2F82,95 ;7387          CLR WR[1]                       ;EXPECTED DATA
U 0813, 9F9D,F5 ;7388          MEM.REQ[READ.MAINT.UBS] ADRS[ZERO] DT[LONG]
                ;7389                                          ;WRITE UNIBUS DATA OF ZEROS
U 0814, 3022,15 ;7390          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0815, 0869,3C ;7391          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0816, 0881,24 ;7392          JMP [LOOP.T6.2]                 ;ERROR LOOP IF ENABLED
                ;7393
U 0817, FF82,15 ;7394          INC LS[ERROR.NUMBER]            ;ERROR 3
U 0818, E5F8,15 ;7395          CLR LS[OS]                      ;CLEAR INDEX
                ;7396  REPEAT.T6.3:
U 0819, 36F7,95 ;7397          MOV LS[SHIFT.OS(4-0)] TO WR[3]  ;GET DATA PATTERN
U 081A, 2341,95 ;7398          ROR WR[3]                       ;CORRECT FOR HARDWARE SHIFT
                ;7399  LOOP.T6.3:
U 081B, 2006,95 ;7400          MOV WR[3] TO WR[1]              ;EXPECTED DATA
U 081C, 9FF7,F5 ;7401          MEM.REQ[READ.MAINT.UBS] ADRS[SHIFT.OS(4-0)] DT[LONG]
                ;7402                                          ;WRITE UNIBUS DATA OF SHIFTING ONES
U 081D, 3022,15 ;7403          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 081E, 0869,3C ;7404          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 081F, 0881,B4 ;7405          JMP [LOOP.T6.3]                 ;ERROR LOOP IF ENABLED
                ;7406
U 0820, 7FF8,15 ;7407          INC LS[OS]                      ;INCREMENT INDEX FOR NEXT PATTERN
U 0821, 36F9,15 ;7408          MOV LS[OS] TO WR[2]             ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                ;7409          BIT LS[BIT4] WITH WR[2],        ;SEE IF OS HAS INCREMENTED TO 10(H)
U 0822, 5949,35 ;7410           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0823, 0881,99 ;7411          JMP.IF[BITS.CLR] TO [REPEAT.T6.3] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                ;7412  END.T6:
```

```
;7413   .PAGE   "  TEST 7 Check MOV MEM.DATA TO LS Logic (M8390  CPU Module)"
;7414   ;++
;7415   ;
;7416   ; Test Description:
;7417   ;
;7418   ;       This test checks the MOV MEM.DATA TO LS logic on the CPU module.  The
;7419   ;       next test must use this function, but it could not be tested until the
;7420   ;       basic logic of the MCT was tested.  The test simply clears an LS loc-
;7421   ;       ation (LS 8), and writes ones in the VAR using the READ.MAINT.VAR
;7422   ;       Memory Function.  It then reads the result with a MOV MEM.DATA TO LS[8]
;7423   ;       instruction, moves LS 8 to WR 0, and checks the result.  The test is
;7424   ;       then repeated using data of zeros.
;7425   ;
;7426   ; Assumptions:
;7427   ;
;7428   ;       It is assumed that all previous tests have run successfully.  This test
;7429   ;       checks logic on the CPU module that must be used in the next test.
;7430   ;
;7431   ; Test Steps:
;7432   ;
;7433   ;       1)  Set up error mask, error number, and module number in LS (for
;7434   ;           error printouts) and clear previous error number in LS.
;7435   ;       2)  Clear LS 8 and issue MEM.REQ with MF=READ.MAINT.VAR and DT=LONG-
;7436   ;           WORD.  Use an LS location containing ones as the VAR data.
;7437   ;       3)  Execute MOV MEM.DATA TO LS[8], then move LS 8 to WR 0.  Check re-
;7438   ;           sult for all ones.
;7439   ;       4)  Repeat steps 2 and 3 with data of zeros.
;7440   ;
;7441   ; Errors:
;7442   ;
;7443   ;       Error 1 - MOV MEM.DATA TO LS failed with data of 1's.
;7444   ;       Error 2 - MOV MEM.DATA TO LS failed with data of 0's.
;7445   ;
;7446   ; Debug:
;7447   ;
;7448   ;       Two methods of debug will be explained.  If the MCT module is in a
;7449   ;       test station then the MCT can be single stepped making debug easier
;7450   ;       in some cases.  Otherwise only the CPU can be single stepped and this
;7451   ;       method will be explained also.  When single stepping the memory con-
;7452   ;       troller will help in debug, this section will explicitly suggest it.
;7453   ;
;7454   ;       Error 1 - This error indicates a problem on the CPU module with the
;7455   ;       DEST CTL ROM or the SRC.ALU CTL PAL.  The LS CONTROL PAL is also a
;7456   ;       possible suspect.  Single step the CPU to the MOV MEM.DATA TO LS
;7457   ;       instruction and check the outputs of the DEST CTL PAL for a 3(B) on
;7458   ;       ALU CTL 2 H through ALU CTL 0 H, and a high on DEST CTL 0 H.  Check
;7459   ;       the outputs of the SRC.ALU CTL PAL for a low on DEST CTL 2 H, a high
;7460   ;       on DEST CTL 1 H and highs on SRC CTL 2 H through SRC CTL 0 H.  If any
;7461   ;       of these are incorrect, suspect the chip that produces them.
;7462   ;         If the ALU control ouputs are OK, check the LS CONTROL PAL for lows
;7463   ;       on output pins 12, 13, and 14.  If any of these are incorrect, suspect
;7464   ;       the LS CONTROL PAL itself
;7465   ;
;7466   ;       Error 2 - Same as error 1.
;7467   ;
```

```
                ;7468   ;--
                ;7469
                ;7470   T.7:
                ;7471
U 0824, B65E,15 ;7472           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;7473                                           ; STATUS WORD
U 0825, 3E80,15 ;7474           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;7475                                           ; BIT 15 INDICATES START OF TEST TO
                ;7476                                           ; CONSOLE
U 0826, 10E0,15 ;7477           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;7478   WAIT.T7.0:
U 0827, 0882,74 ;7479           JMP [WAIT.T7.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                ;7480                                           ; RESPOND
                ;7481
U 0828, 0A1A,AC ;7482           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
U 0829, 3642,15 ;7483           MOV LS[CPU] TO WR[0]            ;SET UP MODULE CODE. BIT 1 SET IN WR
                ;7484
U 082A, 3E8C,15 ;7485           MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                ;7486                                           ; CPU BOARD
                ;7487   LOOP.T7.1:
U 082B, E510,15 ;7488           CLR LS[T8]                      ;COMPLEMENT OF EXPECTED DATA
U 082C, 369E,95 ;7489           MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 082D, 9D9E,75 ;7490           MEM.REQ[READ.MAINT.ADRS] ADRS[ONES] DT[LONG]
                ;7491                                           ;WRITE VAR WITH 1'S
U 082E, 3810,15 ;7492           MOV MEM.DATA TO LS[T8]          ;STORE RESULT IN LS LOCATION 8
U 082F, B610,15 ;7493           MOV LS[T8] TO WR[0]             ;PUT RESULT IN WR 0
U 0830, 0869,3C ;7494           JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0831, 0882,B4 ;7495           JMP [LOOP.T7.1]                 ;ERROR LOOP IF ENABLED
                ;7496
U 0832, FF82,15 ;7497           INC LS[ERROR.NUMBER]            ;ERROR 2
                ;7498   LOOP.T7.2:
U 0833, B69C,95 ;7499           MOV LS[ZERO] TO WR[1]           ;EXPECTED DATA
U 0834, 1D9C,75 ;7500           MEM.REQ[READ.MAINT.ADRS] ADRS[ZERO] DT[LONG]
                ;7501                                           ;WRITE VAR WITH 0'S
U 0835, 3810,15 ;7502           MOV MEM.DATA TO LS[T8]          ;STORE RESULT IN LS LOCATION 8
U 0836, B610,15 ;7503           MOV LS[T8] TO WR[0]             ;PUT RESULT IN WR 0
U 0837, 0869,3C ;7504           JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0838, 0883,34 ;7505           JMP [LOOP.T7.2]                 ;ERROR LOOP IF ENABLED
                ;7506   END.T7:
```

```
;7507   .PAGE   "  TEST 8 Branch On UBC1,ICO, UB ACTIVITY Test (M8391  MCT Module)"
;7508   ;++
;7509   ;
;7510   ; Test Description:
;7511   ;
;7512   ;       This test checks the branch logic for signals L UBC1 H, ICO H, and D
;7513   ;       UB ACTIVITY L.  The signals themselves are also being tested.  These
;7514   ;       signals are controlled by the memory u-code and a branch is performed
;7515   ;       to test them.  If the branch succeeds, then the VAR will be incremented
;7516   ;       by 4.  Reading the VAR at the end of the test will verify that all the
;7517   ;       branches passed, or point to the section that failed first.  Any fail-
;7518   ;       ure will cause an immediate read of the VAR.  CSR 1 is written at the
;7519   ;       beginning of the test to set the DIAG CHK and ECC DIS bits.  These will
;7520   ;       be used to branch to the correct test.  The data path is as follows:
;7521   ;       A MEM.REQ CPU u-instruction is issued with the MF field=RD.MAINT.UBS
;7522   ;       (1F).  The dispatch in memory occurs as described at the beginning of
;7523   ;       this listing.  At the dispatch address, the u-cycle enables UB data
;7524   ;       and UB address, and sets address physical, DATIP, IB BSY and memory
;7525   ;       busy.  The DATIP will set ICO H and clear IC1 H at the start of the
;7526   ;       next u-cycle.  This cycle now branches on DIAG CHK and ECC DIS (set up
;7527   ;       in CSR1 before the MEM.REQ) to the next u-cycle in this test.  This
;7528   ;       next u-cycle sets DATO, keeps IB BSY and memory busy set, keeps UB
;7529   ;       address enabled, and opens control function latch.  Setting DATO will
;7530   ;       clear ICO H and set IC1 H at the start of the next cycle.  Both these
;7531   ;       signals go through the UNIBUS address tranceivers to form UB C0 H and
;7532   ;       UB C1 H which go to the 8 bit latch which is held open by CONT FUNC LAT
;7533   ;       L being high.  These come out as L UBC1 H and L UBC0 H.  In this cycle,
;7534   ;       a branch on ICO H (which is still high from the previous cycle) is per-
;7535   ;       formed to the next u-cycle.  This next u-cycle performs the same func-
;7536   ;       tions as the last cycle except DATO is not set and the VAR is incre-
;7537   ;       mented.  The branch checks for ICO H being low, which occurs as a re-
;7538   ;       sult of the previous DATO.  The next cycle that this one branches to
;7539   ;       performs the same functions as this one except DATI is set and the
;7540   ;       branch is performed on L UBC1.  L UBC1 H is still set from the previous
;7541   ;       set DATO.  Set DATI in this cycle will cause L UBC1 H to go low at the
;7542   ;       start of the next cycle.  The next cycle is a delay cycle to allow L
;7543   ;       UBC1 H to settle before executing a branch on it.  The following cycle
;7544   ;       keeps the CONT FUNC LAT open, keeps memory busy, IB BSY and UB ADR EN
;7545   ;       enabled, increments the VAR and branches on L UBC1 H.  L UBC1 H should
;7546   ;       be low due to the DATI 2 cycles previous.  The next cycle simply keeps
;7547   ;       memory busy set.  The following cycle keeps memory busy set and incre-
;7548   ;       ments the VAR.  This cycle does a branch on D UB ACTIVITY L.  D UB AC-
;7549   ;       TIVITY L becomes active on the second cycle after I BBSY goes away.
;7550   ;       The next 2 cycles are delay cycles which simply keep memory busy as-
;7551   ;       serted.  The delay is needed to allow L BBSY to become deasserted after
;7552   ;       propagating through several levels of logic.  The following cycle will
;7553   ;       keep memory busy asserted, increment the VAR, and branch on D UB ACTIV-
;7554   ;       ITY L, which should now be deasserted (high). The next cycle increments
;7555   ;       the VAR, keeps memory busy asserted, and goes to the READ.MAINT.ADR
;7556   ;       flows to send the current value of the VAR to the CPU for checking.  If
;7557   ;       at any time a branch failure occurs, the READ.MAINT.ADR flow will be
;7558   ;       executed and the value of the VAR at that time will point to the fail-
;7559   ;       ure area.
;7560   ;         This test also checks the input L DTO into the MISC CONTROL PAL to
;7561   ;       assure that it affects the output ICO H.
```

```
;7562    ;
;7563    ; Assumptions:
;7564    ;
;7565    ;       It is assumed that the previous tests have run successfully.
;7566    ;
;7567    ; Test Steps:
;7568    ;
;7569    ;       1)  Set up error number, and module number in LS (for error printouts)
;7570    ;           and clear previous error number in LS.
;7571    ;       2)  Set up error mask to check the low byte and write CSR 1 with data
;7572    ;           to set DIAG CHK and ECC DIS for branching to the correct test.
;7573    ;       3)  Load WR(1) with an C(H) as expected data.
;7574    ;       4)  Issue a MEM.REQ with MF=RD.MAINT.UBS (1F(H)).  Use an LS location
;7575    ;           containing all zeros as the address.  This starts VAR at 0 before
;7576    ;           it increments.  The DT field of MEM.REQ=LONGWORD.
;7577    ;       5)  Execute MOV MEM.DATA TO LS[T8], MOV LS[T8] TO WR[0], and check
;7578    ;           result for a C(H).
;7579    ;       6)  Repeat steps 4 and 5 with data type of BYTE in MEM.REQ and with
;7580    ;           expected data of 2.
;7581    ;
;7582    ; Errors:
;7583    ;
;7584    ;       Error 1 - Branch failure on IC0, L UBC1, or D UB ACTIVITY.  Received
;7585    ;                 data shows where error occured (see debug).
;7586    ;       Error 2 - IC0 H failed to be asserted when data type was BYTE.
;7587    ;
;7588    ; Debug:
;7589    ;
;7590    ;       Two methods of debug will be explained.  If the MCT module is in a
;7591    ;       test station then the MCT can be single stepped making debug easier
;7592    ;       in some cases.  Otherwise only the CPU can be single stepped and this
;7593    ;       method will be explained also.  When single stepping the memory con-
;7594    ;       troller will help in debug, this section will explicitly suggest it.
;7595    ;
;7596    ;       Error 1 - The received data value in the error printout will point to
;7597    ;       the failing branch.  This is the value in the VAR at the point of fail-
;7598    ;       ure.  If VAR = :  0, branch on IC0 H asserted failed to branch;  2,
;7599    ;       branch on IC0 H deasserted failed;  4(H), branch on L UBC1 H asserted
;7600    ;       failed to branch;  6(H), branch on L UBC1 H deasserted failed to
;7601    ;       branch;  8(H) branch on D UB ACTIVITY L asserted failed to branch;
;7602    ;       A(H), branch on D UB ACTIVITY L deasserted failed to branch.
;7603    ;         In the following procedures, if MCT single step is not available, the
;7604    ;       signals will have to be examined while looping on error.
;7605    ;         If VAR=0 then check that setting DATIP asserts IC0 H.  If MCT single
;7606    ;       step is available, single step the CPU to the MEM.REQ instruction and
;7607    ;       step the MCT to the first u-cycle at the dispatch address (which sets
;7608    ;       DATIP).  The inputs to the MISC CONTROL PAL should be high on both SPF0
;7609    ;       H and SPF2 H.  The output will not change until the next cycle.  Step
;7610    ;       to the following cycle (BEN 2=LUB.C0) and check the signal IC0 H for a
;7611    ;       high out of the MISC CONTROL PAL.  If incorrect, the PAL is bad.  If OK
;7612    ;       at the PAL, check IC0 H going into the BEN 2 MUX for a high.  Also
;7613    ;       check the select inputs for a 100(B) on SEL 4,2 and 1 respectively.  If
;7614    ;       these are OK the MUX is probably bad.
;7615    ;         If VAR=2 then check that setting DATO deasserts IC0 H.  If MCT single
;7616    ;       step is available, single step the CPU to the MEM.REQ instruction and
```

```
;7617    ;       step the MCT to the first address after the dispatch address (set
;7618    ;       DATO).  Check the inputs to the MISC CONTROL PAL for a high on SPF2 H,
;7619    ;       a high on SPF1 H, a low on SPF0 H, and a high on L DTO H. L DTO H comes
;7620    ;       from the CPU module's CC CONTROL PAL. It has been checked internal to
;7621    ;       the CPU module in previous CPU tests, but it may not be getting off the
;7622    ;       board to the MCT module.  If these inputs are OK, step the memory to
;7623    ;       the next cycle (another branch with BEN2=LUB.C0) and check the output
;7624    ;       IC0 H for a low.  If OK check IC0 H going into the BEN2 MUX for a low
;7625    ;       and the select lines for a 100(B) on SEL 4,2, and 1 respectively.  If
;7626    ;       all these input are OK, the MUX is probably bad.
;7627    ;         If VAR=4, then check that setting DATO asserts L UBC1 H.  If MCT
;7628    ;       step is available, single step the CPU to the MEM.REQ instruction and
;7629    ;       step the MCT to the first address after the dispatch address (set
;7630    ;       DATO).  Check the inputs to the MISC CONTROL PAL for a high on SPF2 H,
;7631    ;       a high on SPF1 H, and a low on SPF0 H.  If these are OK, single step to
;7632    ;       the next cycle (another BEN2=LUB.C0) and check the output IC1 H for a
;7633    ;       high.  If OK here, check it going into the UNIBUS driver that outputs
;7634    ;       UB C1 H.  Also check UB C1 H for a high and UB ADR EN L for a low.  UB
;7635    ;       C1 H goes to the control function latch.  The input should be high and
;7636    ;       the output, L UBC1 H should be high.  Also check CONT FUNC LAT L for a
;7637    ;       high.  If all these signals are OK, check the input signal SPF2 H into
;7638    ;       the MISC CONTROL PAL for a low (this keeps IC1 H from changing in the
;7639    ;       next cycle) and step the memory controller to the next cycle (BEN1=
;7640    ;       LUB.C1) and check that L UBC1 H into the BEN1 MUX is high.  If not,
;7641    ;       trace it back.  If it has changed at the output of the MISC CONTROL PAL
;7642    ;       (IC1 H) then that PAL is bad.  If it is high, check the select inputs
;7643    ;       for a 100(B) on the SEL 4, 2, and 1 inputs to the BEN1 MUX.  If these
;7644    ;       are OK, suspect the BEN1 MUX itself.
;7645    ;         If VAR=6(H), then check that setting DATI deasserts L UBC1 H.  If MCT
;7646    ;       step is available, single step the CPU to the MEM.REQ instruction and
;7647    ;       step the MCT to the third address after the dispatch address (set
;7648    ;       DATI).  Check the inputs to the MISC CONTROL PAL for a high on SPF2 H
;7649    ;       and a low on SPF1 H.  If these are OK, single step to the next cycle
;7650    ;       (a delay cycle with no branches) and check the output IC1 H for a low.
;7651    ;       If this output is OK, trace it out to L UBC1 H at the output of the
;7652    ;       function control latch, as in the VAR=8 error above, looking for a low
;7653    ;       on C1 signals along the way.  If L UBC1 H is low, check the inputs to
;7654    ;       the MISC CONTROL PAL for a low on SPF2 H (this keeps IC1 H from chang-
;7655    ;       ing on the next clock) and then step to the next cycle (BEN1=LUB.C1).
;7656    ;       Check the inputs to the BEN1 MUX for a low on L UBC1 H and a 100(B) on
;7657    ;       the select inputs SEL 4, 2, and 1 respectively.  If L UBC1 H is high,
;7658    ;       trace it back to the MISC CONTROL PAL.  If IC1 H is high, the PAL is
;7659    ;       bad.  If the inputs to the  BEN1 MUX are OK, suspect the MUX itself.
;7660    ;         If VAR=8(H) then check that dropping IB BSY asserts UB ACTIVITY L.
;7661    ;       If MCT single step is available, step the CPU to the MEM.REQ instruc-
;7662    ;       tion and step the memory u-code to the first u-cycle that branches on
;7663    ;       D UB ACTIVITY L (BEN2=UNIBUS.ACTIVITY.L).  Check the signal D UB ACT-
;7664    ;       IVITY L going into the BEN2 MUX for a low.  The select lines should be
;7665    ;       110 on SEL 4, 2, and 1 respectively.  If these inputs are OK, the MUX
;7666    ;       is probably bad.  If D UB ACTIVITY L is high, then check it at the out-
;7667    ;       put of the latch with UB ACTIVITY L as the input.  If UB ACTIVITY L is
;7668    ;       high, check the MCT ARBITRATOR PAL.  Check the inputs for a low on I
;7669    ;       BBSY H and ARB C0 H, and a high on L BBSY H (pin 4).  If these inputs
;7670    ;       are OK, but UB ACTIVITY L is high, then the MCT ARBITRATOR PAL is bad.
;7671    ;         If L BBSY H is low, it is possible that I BBSY H never went high at
```

```
                ;7672   ;       the MISC CONTROL PAL.  The input I BBSY A H should have been high
                ;7673   ;       throughout this test until 2 cycles previous.  It may be advantagous
                ;7674   ;       to backtrack 2 cycles (by starting over) to check this signal.  If I
                ;7675   ;       BBSY A H goes high but I BBSY H does not, the MISC CONTROL PAL is bad.
                ;7676   ;       If I BBSY H is high, check the inputs to the MISC CONTROL PAL for a low
                ;7677   ;       on IC0 H and a low on I BBSY A H.  If these inputs are OK but I BBSY H
                ;7678   ;       is high, then the MISC CONTROL PAL is bad.
                ;7679   ;         If VAR=A(H) then check that D UB ACTIVITY L goes high 5 cycles after
                ;7680   ;       I BBSY H goes away.  If MCT single step is available, step the CPU to
                ;7681   ;       the MEM.REQ u-instruction and step the memory u-code to the second
                ;7682   ;       cycle that branches on UB ACTIVITY (BEN2=UNIBUS.ACTIVITY.L).  This is
                ;7683   ;       the tenth cycle after the dispatch if all previous branches go the
                ;7684   ;       right way.  Check D UB ACTIVITY L at the BEN2 MUX for a high, and the
                ;7685   ;       select lines for a 110(B) on SEL 4, 2, and 1 respectively.  If these
                ;7686   ;       inputs are OK, the MUX is bad.  If D UB ACTIVITY L is low, check it
                ;7687   ;       at the latch with the input UB ACTIVITY L.  This input should also be
                ;7688   ;       high and comes from the MCT ARBITRATOR PAL.  The inputs to the PAL
                ;7689   ;       should be checked for a low on L NPR H, L SACK H, and L BBSY H and a
                ;7690   ;       high on NPG L.  A defective device on the UNIBUS or a bad UNIBUS could
                ;7691   ;       hold L SACK H or L NPR H high. Also check the latches and TO CLOCK sig-
                ;7692   ;       nals that preceed these inputs.  If all the inputs are OK, suspect the
                ;7693   ;       MCT ARBITRATOR PAL.
                ;7694   ;
                ;7695   ;       Error 2 - This error indicates that IC0 H was not forced high when a
                ;7696   ;       MEM.REQ with a data type of BYTE was executed.  The most likely cause
                ;7697   ;       is in the MISC CONTROL PAL or the input L DTO H.  The input L DTO H
                ;7698   ;       should be low after the MEM.REQ with data type of byte is executed.
                ;7699   ;       If this input is OK, then the PAL is probably bad.
                ;7700   ;--
                ;7701
                ;7702   T.8:
                ;7703
U 0839, B65E,15 ;7704           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;7705                                           ; STATUS WORD
U 083A, 3E80,15 ;7706           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;7707                                           ; BIT 15 INDICATES START OF TEST TO
                ;7708                                           ; CONSOLE
U 083B, 10E0,15 ;7709           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;7710   WAIT.T8.0:
U 083C, 0883,C4 ;7711           JMP [WAIT.T8.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                ;7712                                           ; RESPOND
                ;7713
U 083D, 0A1A,AC ;7714           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
U 083E, B62C,15 ;7715           MOV LS[#FFFFFF00] TO WR[0]      ;SET UP ERROR MASK
U 083F, 3E8A,15 ;7716           MOV WR[0] TO LS[ERROR.MASK]     ;SAVE ERROR MASK TO CHECK THE LOW BYTE
                ;7717
U 0840, 3672,15 ;7718           MOV LS[ECC.DIS] TO WR[0]        ;SET UP ECC DISABLE BIT FOR CSR
U 0841, 4774,15 ;7719           BIS LS[DIAG.CHK] TO WR[0]       ;AND SET DIAG CHK BIT
U 0842, 8A17,FC ;7720           JSR [WRITE.CSR1]                ;SET ECC DIS AND DIAG CHK IN CSR
U 0843, B647,95 ;7721           MOV LS[#8] TO WR[3]             ;GOING TO MAKE AN C(H) FOR EXPECTED DATA
U 0844, 4745,95 ;7722           BIS LS[#4] TO WR[3]             ;WR 3=C(H)
                ;7723   LOOP.T8.1:
U 0845, 2006,95 ;7724           MOV WR[3] TO WR[1]              ;EXPECTED DATA
U 0846, 9F9D,F5 ;7725           MEM.REQ[READ.MAINT.UBS] ADRS[ZERO] DT[LONG]
                ;7726                                           ; START TEST WITH VAR=0
```

```
U 0847, 3810,15 ;7727            MOV MEM.DATA TO LS[T8]          ;READ VALUE OF VAR INTO LS LOCATION
U 0848, B610,15 ;7728            MOV LS[T8] TO WR[0]             ;PUT RECEIVED VALUE IN WR 0
U 0849, 0869,3C ;7729            JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 084A, 8884,54 ;7730            JMP [LOOP.T8.1]                 ;ERROR LOOP IF ENABLED
                ;7731
U 084B, FF82,15 ;7732            INC LS[ERROR.NUMBER]            ;ERROR 2
U 084C, 65FC,15 ;7733            CLR LS[SIZE]                    ;CLEAR SIZE REG TO SIGNIFY BYTE DATA
                ;7734    LOOP.T8.2:
U 084D, B642,95 ;7735            MOV LS[#2] TO WR[1]             ;EXPECTED DATA
U 084E, 9F9D,95 ;7736            MEM.REQ[READ.MAINT.UBS] ADRS[ZERO] DT[BYTE]
                ;7737                                            ; START TEST WITH VAR=0 AND DATA TYPE
                ;7738                                            ; OF BYTE
                ;7739            MOV MEM.DATA TO LS[T8],         ;READ VALUE OF VAR INTO LS LOCATION
U 084F, B810,55 ;7740             DT(SIZE)&SET.ALU.CC            ; KEEP DATA TYPE OF BYTE ENABLED
U 0850, B610,15 ;7741            MOV LS[T8] TO WR[0]             ;PUT RECEIVED VALUE IN WR 0
U 0851, 0869,3C ;7742            JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0852, 0884,D4 ;7743            JMP [LOOP.T8.2]                 ;ERROR LOOP IF ENABLED
                ;7744
                ;7745    END.T8:
```

```
;7746   .PAGE   "  TEST 9 MYSN and SSYN Branch Test (M8391  MCT Module)"
;7747   ;++
;7748   ;
;7749   ; Test Description:
;7750   ;
;7751   ;       This test checks the branch logic for the signals L MSYN H and L SSYN
;7752   ;       H.  The signals themselves are also being tested.  Since MSYN will be
;7753   ;       asserted on the UNIBUS during this test, an address of FFFFFE(H) will
;7754   ;       be sent out with the MSYN to prevent any device that might be on the
;7755   ;       UNIBUS from responding.  The UNIBUS itself however is not being tested.
;7756   ;       The branch on MSYN deasserted is not checked in this test because if it
;7757   ;       didn't work, the dispatch from the memory idle loop would have failed
;7758   ;       in the very first test.
;7759   ;         The data path is as follows:  A write to CSR1 is executed to set
;7760   ;       DIAG CHK and clear ECC DIS.  This will be used to branch to the proper
;7761   ;       test.  A MEM.REQ is issued with the MF field = READ.MAINT.UBS (1F). The
;7762   ;       dispatch occurs as described in the beginning of this listing.  At th
;7763   ;       dispatch address, address physical, UB address and UB data are all
;7764   ;       enabled, IB BSY and bus busy are asserted, and DATIP is set.  A branch
;7765   ;       on ECC DIS and DIAG CHK goes to the correct test.  The next u-cycle
;7766   ;       will issue MSYN and SSYN, keep UB address enabled, and keep IB BSY and
;7767   ;       memory busy asserted.  The following cycle will do the same as the
;7768   ;       previous cycle.  The next cycle will keep UB address enabled, keep IB
;7769   ;       BSY and memory busy asserted, and branch on L MSYN H, checking for MSYN
;7770   ;       being asserted.  Issue SSYN is already propagating through the latches
;7771   ;       at this time.  Issue MSYN and issue SSYN are both dropped.  Dropping
;7772   ;       MSYN will drop SSYN 2 memory cycles later.  The following cycle keeps
;7773   ;       UB address enabled and I BBSY and memory busy asserted.  Also in this
;7774   ;       cycle, a branch on L SSYN H is executed checking for SSYN being assert-
;7775   ;       ed.  The next cycle increments the VAR and keeps UB address, IB BSY and
;7776   ;       memory busy asserted.  The following cycle increments the VAR, keeps
;7777   ;       memory busy asserted, and branches on L SSYN H, checking for SSYN being
;7778   ;       deaserted (SSYN will have gone away by the beginning of this cycle
;7779   ;       since MSYN was dropped 3 cycle previous).  The next cycle increments
;7780   ;       the VAR and goes to the READ.MAINT.ADR flows to read the VAR back to
;7781   ;       the CPU.  If the branches all pass, the VAR will contain B(H) (read as
;7782   ;       5 after shift) when the test has completed.  If a branch failure occurs,
;7783   ;       the memory u-code will go to READ.MAINT.ADDR to read the result.
;7784   ;
;7785   ; Assumptions:
;7786   ;
;7787   ;       It is assumed that all previous tests have run successfully.  It is
;7788   ;       also assumed that the branch on L MSYN H works correctly when MSYN is
;7789   ;       low since this branch is used in the dispatch from the idle loop.
;7790   ;
;7791   ; Test Steps:
;7792   ;
;7793   ;       1)  Set up error number, and module number in LS (for error printouts)
;7794   ;           and clear previous error number in LS.
;7795   ;       2)  Set up error mask to check the low byte only and write CSR1 to set
;7796   ;           DIAG CHK and clear DIS ECC.
;7797   ;       3)  Load WR[1] with 5(H) as expected data.
;7798   ;       4)  Issue MEM.REQ with MF=RD.MAINT.UBS using LS location containing
;7799   ;           FFFFFE for address (so no UNIBUS devices can respond).
;7800   ;       5)  Execute MOV MEM.DATA TO WR[1] and check result for 5(H).
```

```
;7801    ;
;7802    ; Errors:
;7803    ;
;7804    ;       Error 1 - Branch on L MYSN or L SSYN failed.  Received data shows where
;7805    ;                 error occurred (see debug).
;7806    ;
;7807    ; Debug:
;7808    ;
;7809    ;       Two methods of debug will be explained.  If the MCT module is in a
;7810    ;       test station then the MCT can be single stepped making debug easier
;7811    ;       in some cases.  Otherwise only the CPU can be single stepped and this
;7812    ;       method will be explained also.  When single stepping the memory con-
;7813    ;       troller will help in debug, this section will explicitly suggest it.
;7814    ;
;7815    ;       Error 1 - The received data printout in the error report will point to
;7816    ;       the failing branch.  This is the value of the low byte in the VAR at
;7817    ;       the point of failure (after the right shift).  The following is a sum-
;7818    ;       mary of VAR values and corresponding failure description:  VAR=FFFFFE -
;7819    ;       branch on L MSYN asserted failed,  VAR=1(H) - branch on L SSYN asserted
;7820    ;       failed,  VAR=3(H) - branch on L SSYN deasserted failed.
;7821    ;         In the following debug procedures, if MCT single step is not avail-
;7822    ;       able, then the signals will have to be examined while looping on error.
;7823    ;         If VAR=FF, then check that I MSYN H asserts L MSYN H.  If MCT sin-
;7824    ;       gle step is available, step the CPU to the MEM.REQ instruction and step
;7825    ;       the memory controller u-code to the third u-cycle after the dispatch
;7826    ;       address (branch on L MSYN, BEN0=LMSYN).  Check the signal L MSYN H at
;7827    ;       the input to the BEN0 MUX for a high and the select lines for a 111(B)
;7828    ;       on SEL 4, 2, and 1.  If these inputs are OK, the BEN0 MUX is probably
;7829    ;       bad.  If L MSYN H is low, trace it back through the latches.  The clock
;7830    ;       to the latches has already been checked in the previous test.  Trace
;7831    ;       the signal back to I MSYN which comes from the control store PROM.
;7832    ;         If VAR=1(H) then check that I SSYN H asserts L SSYN H.  If MCT single
;7833    ;       step is available, step the CPU to the MEM.REQ instruction and step the
;7834    ;       memory u-code to the third u-cycle after the dispatch address (branch
;7835    ;       on L MSYN, BEN0=LMYSN).  L SSYN H should now be high at the BEN1 MUX,
;7836    ;       although the u-code will not branch on it until the next u-cycle.  If
;7837    ;       L SSYN H is low at the BEN1 MUX, trace it back through the latches to
;7838    ;       the PWR UP AND INIT PAL L ISSYN H.  If this output is low, check the
;7839    ;       input B MSYN H for a high and ENABLE ERROR H for a low.  If these in-
;7840    ;       puts are OK, then the PAL is probably bad.  To verify, start over and
;7841    ;       single step to the second u-cycle after the dispatch address (the one
;7842    ;       immediately preceeding the BEN0=LMSYN cycle) and check the inputs to
;7843    ;       the PWR UP AND INIT PAL for a high on B MSYN H, and I SSYN H, and a low
;7844    ;       on ENABLE ERROR H.  This should force a high on L ISSYN H.  The PAL is
;7845    ;       bad if the inputs are OK.  If the signal L SSYN H is high at the BEN1
;7846    ;       mux, step the u-code to the next u-cycle (BEN1=LSSYN) and check the
;7847    ;       select inputs for a 101(B) on SEL 4, 2, and 1 respectively.  If these
;7848    ;       inputs are OK and L SSYN H is still high, then the MUX is bad.
;7849    ;         If VAR=3(H) then check that deasserting I MSYN H drops I SSYN H.  If
;7850    ;       MCT single step is available, step the CPU to the MEM.REQ u-insrtuction
;7851    ;       and step the memory u-code to the fourth u-cycle after the dispatch
;7852    ;       address (BEN1=LSSYN, no VAR INC).  Check the output L ISSYN H at the
;7853    ;       PWR UP AND INIT PAL for a low.  If this signal is high, check the in-
;7854    ;       puts for a low on B MSYN H and L INTR H.  If these inputs are OK, the
;7855    ;       PAL is bad.  If L ISSYN H is low, step the u-code one more cycle and
```

```
                ;7856   ;       check the output of the first latch, which has B SSYN H as its input,
                ;7857   ;       for a low.  If this is OK, step one more cycle and check the second
                ;7858   ;       latch which outputs L SSYN H for a low.  If OK here check the inputs to
                ;7859   ;       the BEN1 MUX for a low on L SSYN H and a 101(B) on SEL 4, 2, and 1 re-
                ;7860   ;       spectively.  If these inputs are OK, then the MUX is bad.
                ;7861   ;
                ;7862   ;--
                ;7863
                ;7864   T.9:
                ;7865
U 0853, B65E,15 ;7866           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;7867                                           ; STATUS WORD
U 0854, 3E80,15 ;7868           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;7869                                           ; BIT 15 INDICATES START OF TEST TO
                ;7870                                           ; CONSOLE
U 0855, 10E0,15 ;7871           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;7872   WAIT.T9.0:
U 0856, 0885,64 ;7873           JMP [WAIT.T9.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                ;7874                                           ; RESPOND
                ;7875
U 0857, 0A1A,AC ;7876           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
U 0858, B62C,15 ;7877           MOV LS[#FFFFFF00] TO WR[0]      ;SET UP ERROR MASK
U 0859, 3E8A,15 ;7878           MOV WR[0] TO LS[ERROR.MASK]     ;STORE ERROR MASK TO CHECK LOW BYTE
                ;7879
U 085A, 3674,15 ;7880           MOV LS[DIAG.CHK] TO WR[0]       ;SET DIAG CHK BIT TO WRITE IN CSR
U 085B, 8A17,FC ;7881           JSR [WRITE.CSR1]                ;WRITE CSR 1 WITH DATA IN WR[0] (USED
                ;7882                                           ; TO BRANCH INTO TEST)
U 085C, 3645,95 ;7883           MOV LS[#4] TO WR[3]             ;SET UP A 4 IN WR 3
U 085D, 2047,95 ;7884           INC WR[3]                       ;NOW WR 3 CONTAINS A 5 (EXPECTED DATA)
U 085E, DF2A,15 ;7885           MCOM LS[#FF000000] TO WR[0]     ;PUT FFFFFF(H) IN WR 0
U 085F, 2100,15 ;7886           DEC WR[0]                       ;NOW HAVE FFFFFE(H)
U 0860, BE12,15 ;7887           MOV WR[0] TO LS[T9]             ;ADDRESS IN LS
                ;7888   LOOP.T9.1:
U 0861, 2006,95 ;7889           MOV WR[3] TO WR[1]              ;SET UP EXPECTED DATA
U 0862, 1F13,B5 ;7890           MEM.REQ[READ.MAINT.UBS] ADRS[T9] DT[WORD]
                ;7891                                           ;EXECUTE TEST WITH FFFFFE(H) AS UNIBUS
                ;7892                                           ; ADDRESS
U 0863, 3022,15 ;7893           MOV MEM.DATA TO WR[0]           ;GET RESULT IN VAR
U 0864, 0869,3C ;7894           JSR [CHECK.RESULT]              ;CHECK FOR ERROR
U 0865, 8886,14 ;7895           JMP [LOOP.T9.1]                 ;ERROR LOOP IF ENABLED
                ;7896   END.T9:
```

```
;7897    .PAGE   " Data Rotator and ECC Tests"
;7898
;7899    .toc    "  TEST A Write/read Data Through Rotator PAL (M8391  MCT Module)"
;7900    ;++
;7901    ;
;7902    ; Test Description:
;7903    ;
;7904    ;       This test checks the rotator PALs for basic integrity by passing data
;7905    ;       to the rotator and sending the data back without rotation.  This will
;7906    ;       verify the basic operation of the rotator PALs without using the ARRAY
;7907    ;       BUS or ECC latches to store the data.  The data path is as follows: A
;7908    ;       MEM.REQ is issued with MF field=MAINT.ECC.DATA.  The LS address used
;7909    ;       contains data of 0(H) for branching later in the test.  This data is
;7910    ;       loaded into the VAR and a dispatch is made to the proper routine (see
;7911    ;       dispatch description at the beginning of this listing).  At the dis-
;7912    ;       patch address, memory busy and rotate disable are set, the PA bits are
;7913    ;       enabled and transceivers on the CPU module are enabled to allow data
;7914    ;       from the CPU to be received.  In the next cycle, the data rotators are
;7915    ;       clocked to disable rotation and the events of the previous cycle are
;7916    ;       repeated (except SET ADDR PH is not needed).  The following cycle opens
;7917    ;       the ECC latches, enables the output of the data rotators on the ARRAY
;7918    ;       BUS, clears any errors in the CSR, keeps memory busy set, and branches
;7919    ;       on L CPU DR.  CPU DR (data request) comes up when the CPU does a MOV u-
;7920    ;       instruction.  In this case, the CPU is sending the test data pattern
;7921    ;       for the rotator PALs.  If L CPU DR is not yet asserted, memory busy is
;7922    ;       dropped and the memory u-code will loop waiting for it.  Once L CPU DR
;7923    ;       is asserted, the next u-cycle will set 2nd memory cycle, open the ECC
;7924    ;       latches, enable the rotator PAL outputs onto the ARRAY BUS, and keep
;7925    ;       the CPU transceivers enabled.  Setting 2nd memory cycle keeps the data,
;7926    ;       sent over the MC BUS from the CPU, latched inside the rotator PAL (the
;7927    ;       data was loaded in the last u-cycle).  The next cycle keeps the ECC
;7928    ;       latches open, keeps the rotator PAL output on the ARRAY BUS, and waits
;7929    ;       for D DATA REC to be asserted. DATA RCVD is asserted by the CPU at the
;7930    ;       end (P2) of the CPU MOV instruction.  Looping here until the signal is
;7931    ;       asserted before asserting memory busy assures that the CPU will move on
;7932    ;       to the next u-instruction.  When D DATA REC is asserted, the memory u-
;7933    ;       code will branch on LVA00 (which was cleared at the beginning of the
;7934    ;       CPU MEM.REQ instruction) to the proper test.  This u-cycle will assert
;7935    ;       memory busy, open the ECC latches, enable the rotator PAL outputs onto
;7936    ;       the ARRAY BUS, and enable the checkbits from CSR 1 onto the checkbit
;7937    ;       ARRAY BUS.  It will then branch to the proper test according to the
;7938    ;       state of ECC DIS and DIAG CHK in CSR 1.  CSR 1 was loaded prior to
;7939    ;       issueing the MEM.REQ instruction.  For this test, both DIAG CHK and ECC
;7940    ;       DIS are clear.  The next cycle asserts MDR DAT OUT EN, which outputs
;7941    ;       the data pattern latched in the rotator onto the MC BUS, and keeps mem-
;7942    ;       busy asserted.  Now that the data pattern is on the MC BUS, it can be
;7943    ;       read by the CPU, via a MOV MEM.DATA TO WR[0], and checked.  The follow-
;7944    ;       u-cycle drops memory busy, keeps MDR DAT OUT EN asserted, allows the
;7945    ;       output of the ECC latches onto the ARRAY BUS (which is not used in this
;7946    ;       test), and checks for L CPU DR (data request).  If L CPU DR is not yet
;7947    ;       asserted, the u-code loops waiting for it.  The cycle before IDLE loops
;7948    ;       waiting for D DAT REC to become asserted, meaning the CPU has the data.
;7949    ;       The last cycle returns to the IDLE loop.  Data patterns used in this
;7950    ;       test are all 1's, all 0's, and shifted 1's.
;7951    ;
```

```
;7952    ; Assumptions:
;7953    ;
;7954    ;       It is assumed that all previous tests have run successfully.
;7955    ;
;7956    ; Test Steps:
;7957    ;
;7958    ;       1)  Set up error mask, error number, and module number in LS (for
;7959    ;           error printouts) and clear previous error number in LS.
;7960    ;       2)  Write CSR 1 with data to clear both DIAG CHK and ECC DIS.
;7961    ;       3)  Load WR 1 with data pattern from LS.
;7962    ;       4)  Issue a MEM.REQ with DT=longword (11) and MF=MAINT.ECC.DATA using
;7963    ;           a location in LS that contains a 0(H) as the address (this is load-
;7964    ;           ed into the VAR and used to branch later).
;7965    ;       5)  Execute a WRITE.MEM LS with current data pattern.
;7966    ;       6)  Execute five NOPs to allow MC BUS to settle (this assures that the
;7967    ;           rotator PAL is actually driving the MC BUS.
;7968    ;       7)  Execute MOV MEM.DATA TO WR[0] and check result.
;7969    ;       8)  Repeat steps 2 through 6 with next data pattern.
;7970    ;
;7971    ; Errors:
;7972    ;
;7973    ;       Error 1 - Data failed to flow through rotator PAL correctly.  Data=all
;7974    ;                 zeros.
;7975    ;       Error 2 - Data failed to flow through rotator PAL correctly.  Data=all
;7976    ;                 ones.
;7977    ;       Error 3 - Data failed to flow through rotator PAL correctly.  Data=
;7978    ;                 shifting 1's.
;7979    ;
;7980    ; Debug:
;7981    ;
;7982    ;       Two methods of debug will be explained.  If the MCT module is in a
;7983    ;       test station then the MCT can be single stepped making debug easier
;7984    ;       in some cases.  Otherwise only the CPU can be single stepped and this
;7985    ;       method will be explained also.  When single stepping the memory con-
;7986    ;       troller will help in debug, this section will explicitly suggest it.
;7987    ;
;7988    ;       This test uses a new branch control signal that could cause the memory
;7989    ;       u-code to hang waiting for D DATA REC H.  If the test reports a timeout
;7990    ;       at the WRITE.MEM instruction, check the signal D DATA REC H at the BEN
;7991    ;       MUX for a high.  The select inputs should be 110(B) on SEL 4, 2, and 1
;7992    ;       respectively.  If these are OK, the MUX is probably bad.  If D DATA REC
;7993    ;       H is low, check it at the latch that has DATA RCVD H as its input.  The
;7994    ;       signal DATA RCVD H has been checked as an input to the MCT ARBITRATOR
;7995    ;       PAL but never as an input to this latch.
;7996    ;
;7997    ;       Error 1 - This error could be caused by a bad rotator PAL or a multi-
;7998    ;       tude of control signals.  Single step the CPU to the MEM.REQ instruc-
;7999    ;       tion and check the signals at the data rotators for the following (if
;8000    ;       all bits are not failing, pick a PAL that outputs one that is failing):
;8001    ;       2ND MEM CYCLE H should be low,  MDR DAT OUT EN L should be high, A0 L
;8002    ;       and A1 L should both be high, and T0 CLOCK (2) H should be clocking
;8003    ;       (this u-cycle is looping on itself).  If 2ND MEM CYC H is high, check
;8004    ;       it at the output of the MISC CONTROL PAL.  If incorrect there check the
;8005    ;       input for a low on SPF1 H.  The output is initially forced low in the
;8006    ;       IDLE loop.  If the output is high, go back to the idle loop and check
```

```
                ;8007   ;       for CONT FUNC LAT L being high.  If this is OK, the output should be
                ;8008   ;       low, otherwise the PAL is bad.  SPF1 should stay low up to where 2nd
                ;8009   ;       memory cycle is set, and the output should not go high until then.  If
                ;8010   ;       A1 or A0 are low, check the DATA ROTATOR CONTROL PAL.  The signal ROT
                ;8011   ;       CLOCK H should be low.  If this input is OK, but the output A1 or A0 is
                ;8012   ;       low, the inputs at the time of ROT CLOCK will have to be checked.  If
                ;8013   ;       MCT single step is available, step the mem u-code to the address after
                ;8014   ;       the dispatch (with ROT CLOCK) and check for a high on ROT C1 H, ROT C0
                ;8015   ;       H, CPH/UBL H and ROT CLOCK H.  If these are all OK, the PAL is probably
                ;8016   ;       bad.  CPH/UBL comes from the CONTROL PREFETCH PAL.  If CPH/UBL is low,
                ;8017   ;       check CPU GRANT L for a low and CONT FUNC LAT L for a high while loop-
                ;8018   ;       ing on the memory u-cycle that checks for L CPU DR H (single step CPU
                ;8019   ;       to MEM.REQ and let memory run to loop on this u-cycle).
                ;8020   ;         If the inputs to the data rotators are OK, single step the CPU to the
                ;8021   ;       WRITE.MEM instruction.  The memory u-code will be looping waiting for
                ;8022   ;       CPU DATA RCVD to be asserted.  The outputs on the MC BUS can now be
                ;8023   ;       checked and should be the expected data.  The inputs should also be
                ;8024   ;       checked for high on 2ND MEMORY CYCLE H.  2ND MEMORY CYCLE H is set in
                ;8025   ;       the previous cycle by asserting SPF1 H high.  If 2ND MEMORY CYCLE H is
                ;8026   ;       low at the MISC CONTROL PAL, and MCT single step is available, then
                ;8027   ;       start at the MEM.REQ instruction again, enable MCT single step, step
                ;8028   ;       the CPU to the WRITE.MEM instruction, and step the memory u-code to the
                ;8029   ;       next u-cycle (sets 2ND MEMORY CYCLE H).  Check the input SPF1 H into
                ;8030   ;       the MISC CONTROL PAL for a high.  The output should be high.  If not
                ;8031   ;       the PAL is bad.  Step the u-code once more and check the inputs CONT
                ;8032   ;       FUNC LAT L for a low and SPF0 H, SPF1 H, and SPF2 H for lows.  If 2ND
                ;8033   ;       MEMORY CYCLE H goes low now, the PAL is bad.
                ;8034   ;         If 2ND MEMORY CYCLE was OK into the rotator PAL, but the outputs
                ;8035   ;       were incorrect, check the inputs by single stepping the CPU to the
                ;8036   ;       MEM.REQ instruction.  The memory u-code will be looping waiting for
                ;8037   ;       CPU DATA REQ.  The inputs on the MC BUS can be checked during this
                ;8038   ;       loop.
                ;8039   ;         Single step the CPU to the MOVE MEM.DATA TO WR instruction and let
                ;8040   ;       the memory loop on the next to last u-cycle in the test, waiting for
                ;8041   ;       CPU DATA RCVD to be asserted.  Check the inputs to the data rotator
                ;8042   ;       PALs for a low on MDR DATA OUT EN L and a high on 2ND MEMORY CYCLE H.
                ;8043   ;       The output on the MC BUS should be the expected data.  If the inputs
                ;8044   ;       are OK, the rotator PAL is probably bad (a remote possibility is that
                ;8045   ;       the control store failed to output the correct bits in some previous
                ;8046   ;       cycle).
                ;8047   ;
                ;8048   ;       Error 2 - Same as error 1.  Only the data pattern is different.
                ;8049   ;
                ;8050   ;       Error 3 - See error 1, but the rotator PAL itself is the most likely
                ;8051   ;       suspect.
                ;8052   ;
                ;8053   ;--
                ;8054
                ;8055   T.A:
                ;8056
U 0866, B65E,15 ;8057           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;8058                                           ; STATUS WORD
U 0867, 3E80,15 ;8059           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;8060                                           ; BIT 15 INDICATES START OF TEST TO
                ;8061                                           ; CONSOLE
```

```
U 0868, 10E0,15 ;8062            MISC [SET.CP.ATTN]                ;ISSUE CPU ATTN TO CONSOLE
                ;8063    WAIT.TA.0:
U 0869, 0886,94 ;8064            JMP [WAIT.TA.0]                   ;LOOP HERE WAITING FOR CONSOLE TO
                ;8065                                              ; RESPOND
                ;8066
U 086A, 0A1A,9C ;8067            JSR [SETUP]                       ;SET UP MASKS, MODULE CODE ETC. AND
                ;8068                                              ; CLEAR CSR 1.
                ;8069    LOOP.TA.1:
U 086B, B69C,95 ;8070            MOV LS[ZERO] TO WR[1]             ;EXPECTED DATA
U 086C, 9D9C,F5 ;8071            MEM.REQ[MAINT.ECC.DATA] ADRS[ZERO] DT[LONG]
                ;8072                                              ;EXECUTE DATA ROTATOR TEST
U 086D, B29C,15 ;8073            WRITE.MEM LS[ZERO]                ;WRITE ZEROS IN ROTATOR
U 086E, 0A1B,4C ;8074            JSR [NOP.DELAY]                   ;EXECUTE 5 CYCLE DELAY TO ALLOW MC BUS
                ;8075                                              ; TO SETTLE
U 086F, 3022,15 ;8076            MOV MEM.DATA TO WR[0]             ;GET RESULT FROM DATA ROTATORS
U 0870, 0869,3C ;8077            JSR [CHECK.RESULT]                ;CHECK FOR ERROR
U 0871, 8886,B4 ;8078            JMP [LOOP.TA.1]                   ;ERROR LOOP IF ENABLED
                ;8079
U 0872, FF82,15 ;8080            INC LS[ERROR.NUMBER]              ;ERROR 2
                ;8081    LOOP.TA.2:
U 0873, 369E,95 ;8082            MOV LS[ONES] TO WR[1]             ;EXPECTED DATA
U 0874, 9D9C,F5 ;8083            MEM.REQ[MAINT.ECC.DATA] ADRS[ZERO] DT[LONG]
                ;8084                                              ;EXECUTE DATA ROTATOR TEST
U 0875, 329E,15 ;8085            WRITE.MEM LS[ONES]                ;WRITE ONES IN ROTATOR
U 0876, 0A1B,4C ;8086            JSR [NOP.DELAY]                   ;EXECUTE 5 CYCLE DELAY TO ALLOW MC BUS
                ;8087                                              ; TO SETTLE
U 0877, 3022,15 ;8088            MOV MEM.DATA TO WR[0]             ;GET RESULT FROM DATA ROTATORS
U 0878, 0869,3C ;8089            JSR [CHECK.RESULT]                ;CHECK FOR ERROR
U 0879, 8887,34 ;8090            JMP [LOOP.TA.2]           ;ERROR LOOP IF ENABLED
                ;8091
U 087A, FF82,15 ;8092            INC LS[ERROR.NUMBER]              ;ERROR 3
U 087B, E5F8,15 ;8093            CLR LS[OS]                        ;CLEAR INDEX
                ;8094    LOOP.TA.3:
U 087C, B6F6,95 ;8095            MOV LS[SHIFT.OS(4-0)] TO WR[1]    ;GET NEXT DATA PATTERN (EXPECTED DATA)
U 087D, 9D9C,F5 ;8096            MEM.REQ[MAINT.ECC.DATA] ADRS[ZERO] DT[LONG]
                ;8097                                              ;EXECUTE DATA ROTATOR TEST
U 087E, B2F6,15 ;8098            WRITE.MEM LS[SHIFT.OS(4-0)]       ;WRITE SHIFTING ONES IN ROTATOR
U 087F, 0A1B,4C ;8099            JSR [NOP.DELAY]                   ;EXECUTE 5 CYCLE DELAY TO ALLOW MC BUS
                ;8100                                              ; TO SETTLE
U 0880, 3022,15 ;8101            MOV MEM.DATA TO WR[0]             ;GET RESULT FROM DATA ROTATORS
U 0881, 0869,3C ;8102            JSR [CHECK.RESULT]                ;CHECK FOR ERROR
U 0882, 8887,C4 ;8103            JMP [LOOP.TA.3]                   ;ERROR LOOP IF ENABLED
                ;8104
U 0883, 7FF8,15 ;8105            INC LS[OS]                        ;INCREMENT INDEX FOR NEXT PATTERN
U 0884, 36F9,15 ;8106            MOV LS[OS] TO WR[2]               ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                ;8107            BIT LS[BIT5] WITH WR[2],          ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0885, D94B,35 ;8108             DT(LONG)&SET.ALU.CC              ; AND SET CONDITION CODES
U 0886, 0887,C9 ;8109            JMP.IF[BITS.CLR] TO [LOOP.TA.3] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                ;8110    END.TA:
```

```
;8111    .PAGE   "  TEST B ARY BUS, ECC Latches, and CSR CKBTS Test (M8391  MCT Module)"
;8112    ;++
;8113    ;
;8114    ; Test Description:
;8115    ;
;8116    ;       This test checks the ECC latches and ARRAY BUS by outputting data from
;8117    ;       the data rotators onto the ARRAY BUS and latching the data in the ECC
;8118    ;       chip.  The data is then read back over the same path and checked.  CSR
;8119    ;       1, CSR 0, and the ECC CKBT latches are also tested.  CSR 1 is written
;8120    ;       with checkbits and the test writes these CKBTs into the ECC CB latch.
;8121    ;       Then the CKBTs from the ECC CB latch are put on the CB BUS and written
;8122    ;       into CSR 0 where it can be checked.
;8123    ;         The data path is as follows:  A MEM.REQ is issued with MF field=MAINT
;8124    ;       .ECC.DATA.  The LS address used contains data of 0(H) for branching
;8125    ;       later in the test.  This data is loaded into the VAR and a dispatch is
;8126    ;       made to the proper routine (see dispatch description at the beginning
;8127    ;       of this listing).  At the dispatch address, memory busy and rotate
;8128    ;       disable are set, the PA bits are enabled and transceivers on the CPU
;8129    ;       module are enabled to allow data from the CPU to be received.  In the
;8130    ;       next cycle, the data rotators are clocked to disable rotation and the
;8131    ;       events of the previous cycle are repeated (except SET ADDR PH is not
;8132    ;       needed).  The following cycle opens the ECC latches, enables the output
;8133    ;       of the data rotators on the ARRAY BUS, clears any errors in the CSR,
;8134    ;       keeps memory busy set, and branches on L CPU DR.
;8135    ;         CPU DR (data request) comes up when the CPU does a MOV u-instruction.
;8136    ;       In this case, the CPU is sending the test data pattern for the rotator
;8137    ;       PALs.  If L CPU DR is not yet asserted, memory busy is dropped and the
;8138    ;       memory u-code will loop waiting for it.  Once L CPU DR is asserted, the
;8139    ;       next u-cycle will set 2nd memory cycle, open the ECC latches, enable
;8140    ;       the rotator PAL outputs onto the ARRAY BUS, and keep the CPU trans-
;8141    ;       ceivers enabled.  Setting 2nd memory cycle keeps the data, sent over
;8142    ;       the MC BUS from the CPU, latched inside the rotator PAL (the data was
;8143    ;       loaded in the last u-cycle).  The next cycle keeps the ECC latches
;8144    ;       open, keeps the rotator PAL output on the ARRAY BUS, and waits for D
;8145    ;       DATA REC to be asserted. DATA RCVD is asserted by the CPU at the
;8146    ;       end (P2) of the CPU MOV instruction.  Looping here until the signal is
;8147    ;       asserted before asserting memory busy assures that the CPU will move on
;8148    ;       to the next u-instruction.
;8149    ;         When D DATA REC is asserted, the memory u-code will branch on LVA00
;8150    ;       (which was cleared at the beginning of the CPU MEM.REQ instruction) to
;8151    ;       the proper test.  This u-cycle will assert memory busy, open the ECC
;8152    ;       latches, enable the rotator PAL outputs onto the ARRAY BUS, and enable
;8153    ;       the checkbits from CSR 1 onto the checkbit ARRAY BUS.  It will then
;8154    ;       branch to the proper test according to the state of ECC DIS and DIAG
;8155    ;       CHK in CSR 1.  CSR 1 was loaded prior to issueing the MEM.REQ instruc-
;8156    ;       tion. For this test, DIAG CHK is set and ECC DIS is set, and the CHKBTS
;8157    ;       are written in the lower 7 bits of CSR 1.  The following u-cycle will
;8158    ;       close the ECC data in, data out and CKBT latches.  2ND MEMORY CYCLE is
;8159    ;       cleared so that the data rotators can take the data off the ARRAY BUS
;8160    ;       later.  Memory busy is left asserted.  The next u-cycle asserts the
;8161    ;       syndrome bits on the CB BUS (to assure that the CKBT latches are clos-
;8162    ;       ing) and keeps memory busy asserted.  The following u-cycle outputs the
;8163    ;       latched CKBTS onto the CB BUS, keeps memory busy asserted, and branches
;8164    ;       on ECC error.  If an error was detected (the CKBTS do not match those
;8165    ;       generated for the data), then the u-code will branch to an instruction
```

```
;8166    ;       which asserts CSR ERR ADDR CLK to the CSR CONTROL PAL.  The RDS or CRD
;8167    ;       bits in CSR 1 will be set, depending on whether a single error, or
;8168    ;       multiple error was detected.  In addition, DATA ERR will be asserted
;8169    ;       which asserts ERR SUM to the CPU.  The latched CKBTS from the ECC chip
;8170    ;       are kept enabled on the CB BUS.  If no error is detected, CSR ERROR
;8171    ;       ADDR CLK is not asserted and no error sum, RDS, or CRD will be report-
;8172    ;       ed.
;8173    ;         The following u-cycle enables the ECC data onto the ARRAY BUS and en-
;8174    ;       ables the data rotators to output this data on the MC BUS.  It also
;8175    ;       clocks CSR 0 with the CKBTS which were latched in the ECC chip.  Put-
;8176    ;       ting the ECC data on the ARRAY BUS and enabling the data rotators to
;8177    ;       output the data on the MC BUS is repeated in the next cycle.  Now that
;8178    ;       the data pattern is on the MC BUS, it can be read by the CPU, via a MOV
;8179    ;       MEM.DATA TO WR[0], and checked.  The folowing cycle drops memory busy,
;8180    ;       keeps MDR DAT OUT EN asserted, allows the output of the ECC latches
;8181    ;       onto the ARRAY BUS, and checks for L CPU DR (data request).  If L CPU
;8182    ;       DR is not yet asserted, the u-code loops waiting for it.  The cycle
;8183    ;       before IDLE loops waiting for D DAT REC to become asserted, meaning the
;8184    ;       CPU has the data.  The last cycle returns to the IDLE loop.
;8185    ;         Data patterns used in this test are as follows:  For ECC data - all
;8186    ;       1's, all 0's, and shifting 1's.  For ECC CKBTS - all 0's, all 1's, and
;8187    ;       shifting 1's.
;8188    ;
;8189    ; Assumptions:
;8190    ;
;8191    ;       It is assumed that all previous tests have run successfully.
;8192    ;
;8193    ; Test Steps:
;8194    ;
;8195    ;       1)  Set up error mask, error number, and module number in LS (for
;8196    ;           error printouts) and clear previous error number in LS.
;8197    ;       2)  Write CSR 1 with data to set both DIAG CHK and ECC DIS.  Also load
;8198    ;           CKBT portion (bits 0-6) with test pattern for CKBT test.
;8199    ;       3)  Load WR 1 with data pattern from LS (ECC data expected result).
;8200    ;       4)  Issue a MEM.REQ with DT=longword (11) and MF=MAINT.ECC.DATA using
;8201    ;           a location in LS that contains a 0(H) as the address (this is load-
;8202    ;           ed into the VAR and used to branch later).
;8203    ;       5)  Execute a WRITE.MEM LS with current data oattern.
;8204    ;       6)  Execute MOV MEM.DATA TO WR[0] and check result of ECC data.
;8205    ;       7)  Change error mask to check bits 0-6 and read CSR 0 to check CKBT
;8206    ;           data.
;8207    ;       8)  Restore error mask to 32 bits and repeat steps 2 through 7 with
;8208    ;           next data pattern.
;8209    ;
;8210    ; Errors:
;8211    ;
;8212    ;       Error 1 - ECC latch, ARRAY BUS, or rotator failed with data of all 1's.
;8213    ;       Error 2 - CKBT latch, CKBT BUS, CSR 1, or CSR 0 failed with data of all
;8214    ;                 0's.
;8215    ;       Error 3 - ECC latch, ARRAY BUS, or rotator failed with data of all 0's.
;8216    ;       Error 4 - CKBT latch, CKBT BUS, CSR 1, or CSR 0 failed with data of all
;8217    ;                 1's.
;8218    ;       Error 5 - ECC latch, ARRAY BUS, or rotator failed with data of shifting
;8219    ;                 1's.
;8220    ;       Error 6 - CKBT latch, CKBT BUS, CSR 1, or CSR 0 failed with data of
```

```
                ;8221   ;               shifting 1's.
                ;8222   ;
                ;8223   ; Debug:
                ;8224   ;
                ;8225   ;       Two methods of debug will be explained.  If the MCT module is in a
                ;8226   ;       test station then the MCT can be single stepped making debug easier
                ;8227   ;       in some cases.  Otherwise only the CPU can be single stepped and this
                ;8228   ;       method will be explained also.  When single stepping the memory con-
                ;8229   ;       troller will help in debug, this section will explicitly suggest it.
                ;8230   ;
                ;8231   ;       Error 1 - This test is the first to use the ARRAY BUS or the ECC chip.
                ;8232   ;       It also uses a new path through the data rotator PAL.  Check the output
                ;8233   ;       of the data rotator and control signals to the ECC chip first by single
                ;8234   ;       stepping the CPU to the WRITE.MEM instruction.  The memory u-code will
                ;8235   ;       now loop waiting for CPU DATA RCVD.  Check the output of the data rot-
                ;8236   ;       ator for lows on BUS ARRAY Dxx L.  If incorrect, check DIR WR BYT EN L
                ;8237   ;       on pin 11 for a low.  This comes directly from the control store PROM.
                ;8238   ;       If the enable is low, the PAL is probably bad (the other logic was used
                ;8239   ;       in the previous test).  If the outputs on BUS ARRAY Dxx L are OK, check
                ;8240   ;       the ARRAY BUS inputs to the ECC chips themselves.  Also check LAT DAT
                ;8241   ;       IN H for a low (pin 1 of both ECC chips) and LAT OUPUT BYT x L for a
                ;8242   ;       high (pins 3 and 45 of both ECC chips).  LAT OUTPUT BYT x L comes from
                ;8243   ;       the negated input 'AND' gates with LAT OUTPUT BYT L from the control
                ;8244   ;       store prom and the output of the MDR AND DIR CONTROL PAL as its inputs.
                ;8245   ;       The MDR AND DIR CONTROL PAL should be outputting a low on pins 12, 13,
                ;8246   ;       14, and 15 which enable one side of the negated input 'AND' gates.  If
                ;8247   ;       one of these is high, check the input ROT C0 H for a low.  This is all
                ;8248   ;       that is necessary to force the outputs low.  If ROT C0 H is OK, the PAL
                ;8249   ;       is bad.
                ;8250   ;         If all's well at this point, single step the CPU to the MOV MEM.DATA
                ;8251   ;       TO WR[0] instruction.  The memory will be looping on the next to last
                ;8252   ;       u-cycle waiting for CPU DATA RCVD.  Check the ARRAY BUS Dxx L signals
                ;8253   ;       into the data rotator PALs.  They should all be low, being driven by
                ;8254   ;       the ECC chips.  If the ARRAY BUS inputs are incorrect, check the ECC
                ;8255   ;       chip inputs for the following:  LAT OUTPUT BYT x L should be low,  LAT
                ;8256   ;       DAT IN H should be high, and OUTPUT BYT (0-3) H should be high.  OUTPUT
                ;8257   ;       BYTE and LAT DAT IN come directly from the control store PROM, while
                ;8258   ;       LAT OUTPUT BYT x L comes from the negated input 'AND' gates.  If this
                ;8259   ;       signal is wrong, check that LAT OUTPUT BYT L (C63) into the 'AND' gates
                ;8260   ;       is high.  If these inputs are OK, but the output is incorrect, suspect
                ;8261   ;       the ECC chip itself.
                ;8262   ;         If the input to the data rotators on the ARRAY BUS is correct, check
                ;8263   ;       the other inputs to the rotator PAL for the following:  2ND MEM CYC H
                ;8264   ;       should be low, A0 L and A1 L should be high, MDR DAT OUT EN L should be
                ;8265   ;       low.  If these inputs are OK, the outputs on BUS MC Dxx H should be
                ;8266   ;       high.  If not, the rotator PAL is bad.
                ;8267   ;         Another possibility is that one of the array cards is holding the BUS
                ;8268   ;       ARRAY Dxx lines or shorting 2 together.  Remove any arrays and run this
                ;8269   ;       test again to check for this problem.
                ;8270   ;
                ;8271   ;       Error 2 - This error indicates a problem with bits 0-6 of CSR 1 or CSR
                ;8272   ;       0, or a problem with the CKBT BUS, ECC CKBT latches, or control lines.
                ;8273   ;       Single step the CPU to the WRITE.MEM instruction.  The memory u-code
                ;8274   ;       will then be looping waiting for CPU DAT RCVD.  Check the outputs on
                ;8275   ;       the CSR 1 latch (BUS ARRAY CB x L) for all lows.  If incorrect, check
```

```
                    ;8276   ;       CSR CB EN L (C43) for a low.  If this is OK, check the other inputs for
                    ;8277   ;       floats (no connection).  If no floats are found, the WRITE CSR u-flow
                    ;8278   ;       must be run to check the clock and BUS MC inputs.  If MCT single step
                    ;8279   ;       is not available, loop on test must be used.
                    ;8280   ;         If the outputs on BUS ARRAY CB x L are OK, check the signal LAT CB
                    ;8281   ;       REG L for a high at the ECC chip and check the BUS ARRAY CB x L inputs
                    ;8282   ;       at the ECC chip for lows.
                    ;8283   ;         If all is OK up to here, and MCT single step is available, set up the
                    ;8284   ;       memory for single step and step the CPU to the READ.MEM u-instruction.
                    ;8285   ;       Step the memory controller to the u-cycle that outputs the CB BUS and
                    ;8286   ;       clocks CSR 0 (this is the cycle that follows the branch on error, BEN
                    ;8287   ;       0=ERROR.L, if there were no error).  Check the BUA ARRAY CB x L lines
                    ;8288   ;       going to the CSR 0 latch for all lows.  Also check the signal CSR CB/
                    ;8289   ;       SYN CLK H for a high at the CSR 0 latch.  If these signals are OK, then
                    ;8290   ;       either the CSR 0 latch is bad, the CSR CB/SYN CLK is not going low, or
                    ;8291   ;       the RD CSR 0 L signal is not going low to enable the latch when READ
                    ;8292   ;       CSR is executed.  If the BUS ARRAY CB x L signals are incorrect, check
                    ;8293   ;       that the signal OUTPUT CB BUS H (C26) at the ECC chip is high, and that
                    ;8294   ;       LAT CB REG L is low.  If these are OK, suspect the ECC chip.
                    ;8295   ;
                    ;8296   ;       Error 3 - Same as error 1 except data is all 0's.
                    ;8297   ;
                    ;8298   ;       Error 4 - Same as error 2 except data is all 1's.
                    ;8299   ;
                    ;8300   ;       Error 5 - Suspect first a short between two ARRAY BUS lines or an i. -
                    ;8301   ;       ternal short within the ECC chip or data rotator PAL.
                    ;8302   ;
                    ;8303   ;       Error 6 - Suspect first a short between two CKBT BUS lines or an in-
                    ;8304   ;       ternal short within the ECC chip, CSR 1 latch or CSR 0 latch.
                    ;8305   ;
                    ;8306   ;--
                    ;8307
                    ;8308   T.B:
                    ;8309
U 0887, B65E,15     ;8310           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;8311                                           ; STATUS WORD
U 0888, 3E80,15     ;8312           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;8313                                           ; BIT 15 INDICATES START OF TEST TO
                    ;8314                                           ; CONSOLE
U 0889, 10E0,15     ;8315           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;8316   WAIT.TB.0:
U 088A, 8888,A4     ;8317           JMP [WAIT.TB.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                    ;8318                                           ; RESPOND
                    ;8319
U 088B, 0A1A,AC     ;8320           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
U 088C, B713,95     ;8321           MOV LS[#FFFFFF80] TO WR[3]      ;SET UP ERROR MASK FOR 7 BITS (6-0)
                    ;8322                                           ; FOR EVEN NUMBERED ERRORS
                    ;8323
U 088D, 3641,15     ;8324           MOV LS[#1] TO WR[2]             ;1 IN WR 2 (USED TO SET ERROR NUMBER)
                    ;8325
U 088E, 3672,15     ;8326           MOV LS[ECC.DIS] TO WR[0]        ;SET ECC DISABLE BIT IN WR
U 088F, 4774,15     ;8327           BIS LS[DIAG.CHK] TO WR[0]       ;ALSO SET DIAG CHK BIT
U 0890, 8A17,FC     ;8328           JSR [WRITE.CSR1]                ;SET ECC DIS AND DIAG CHK BITS IN CSR 1
                    ;8329                                           ; CLEAR CKBT PATTERN IN BITS 6-0
                    ;8330   LOOP.TB.1:
```

```
U 0891, E58A,15 ;8331          CLR LS[ERROR.MASK]                ;CHECK ALL BITS
U 0892, 3E83,15 ;8332          MOV WR[2] TO LS[ERROR.NUMBER]     ;SET UP ERROR NUMBER IN LS (ERROR 1)
U 0893, 369E,95 ;8333          MOV LS[ONES] TO WR[1]             ;EXPECTED DATA FOR DATA BITS
U 0894, 9D9C,F5 ;8334          MEM.REQ[MAINT.ECC.DATA] ADRS[ZERO] DT[LONG]
                ;8335                                            ;EXECUTE ECC CHIP DATA TEST
U 0895, 329E,15 ;8336          WRITE.MEM LS[ONES]                ;WRITE DATA OF ONES IN ECC CHIP.  CKBTS
                ;8337                                            ; GET DATA OF 0 FROM CSR 1
U 0896, 3022,15 ;8338          MOV MEM.DATA TO WR[0]             ;GET DATA BITS RESULT (SHOULD BE ONES)
U 0897, 0869,3C ;8339          JSR [CHECK.RESULT]                ;CHECK FOR ERROR
U 0898, 8889,14 ;8340          JMP [LOOP.TB.1]                   ;ERROR LOOP IF ENABLED
                ;8341
U 0899, 3E8B,95 ;8342          MOV WR[3] TO LS[ERROR.MASK]       ;STORE MASK IN LS
U 089A, FF82,15 ;8343          INC LS[ERROR.NUMBER]              ;ERROR 2
U 089B, B69C,95 ;8344          MOV LS[ZERO] TO WR[1]             ;EXPECTED DATA
U 089C, 9D9D,75 ;8345          MEM.REQ[READ.CSR] ADRS[#0] DT[LONG]
                ;8346                                            ;READ CSR 0 FOR CKBTS
U 089D, 3022,15 ;8347          MOV MEM.DATA TO WR[0]             ;GET RESULT OF CKBTS
U 089E, 0869,3C ;8348          JSR [CHECK.RESULT]                ;CHECK FOR ERRORS
U 089F, 8889,14 ;8349          JMP [LOOP.TB.1]                   ;ERROR LOOP IF ENABLED
                ;8350
U 08A0, 4043,15 ;8351          ADD LS[#2] TO WR[2]               ;ERROR 3 IN WR 2
                ;8352
U 08A1, B626,15 ;8353          MOV LS[#FF] TO WR[0]              ;SET BITS 7-0 IN WR
U 08A2, 4772,15 ;8354          BIS LS[ECC.DIS] TO WR[0]          ;SET ECC DISABLE BIT IN WR
U 08A3, 4774,15 ;8355          BIS LS[DIAG.CHK] TO WR[0]         ;ALSO SET DIAG CHK BIT
U 08A4, 8A17,FC ;8356          JSR [WRITE.CSR1]                  ;SET ECC DIS AND DIAG CHK BITS IN CSR 1
                ;8357                                            ; SET CKBT PATTERN IN BITS 6-0 (7 IS
                ;8358                                            ; ALSO SET BUT IS NOT USED IN CSR 1)
                ;8359  LOOP.TB.3:
U 08A5, E58A,15 ;8360          CLR LS[ERROR.MASK]                ;CHECK ALL BITS
U 08A6, 3E83,15 ;8361          MOV WR[2] TO LS[ERROR.NUMBER]     ;SET UP ERROR NUMBER IN LS (ERROR 3)
U 08A7, B69C,95 ;8362          MOV LS[ZERO] TO WR[1]             ;EXPECTED DATA FOR DATA BITS
U 08A8, 9D9C,F5 ;8363          MEM.REQ[MAINT.ECC.DATA] ADRS[ZERO] DT[LONG]
                ;8364                                            ;EXECUTE ECC CHIP DATA TEST
U 08A9, B29C,15 ;8365          WRITE.MEM LS[ZERO]                ;WRITE DATA OF ZEROS IN ECC CHIP.  CKBTS
                ;8366                                            ; GET DATA OF 1'S FROM CSR 1
U 08AA, 3022,15 ;8367          MOV MEM.DATA TO WR[0]             ;GET DATA BITS RESULT (SHOULD BE ZEROS)
U 08AB, 0869,3C ;8368          JSR [CHECK.RESULT]                ;CHECK FOR ERROR
U 08AC, 088A,54 ;8369          JMP [LOOP.TB.3]                   ;ERROR LOOP IF ENABLED
                ;8370
U 08AD, 3E8B,95 ;8371          MOV WR[3] TO LS[ERROR.MASK]       ;STORE MASK IN LS
U 08AE, FF82,15 ;8372          INC LS[ERROR.NUMBER]              ;ERROR 4
U 08AF, 369E,95 ;8373          MOV LS[ONES] TO WR[1]             ;EXPECTED DATA
U 08B0, 9D9D,75 ;8374          MEM.REQ[READ.CSR] ADRS[#0] DT[LONG]
                ;8375                                            ;READ CSR 0 FOR CKBTS
U 08B1, 3022,15 ;8376          MOV MEM.DATA TO WR[0]             ;GET RESULT OF CKBTS
U 08B2, 0869,3C ;8377          JSR [CHECK.RESULT]                ;CHECK FOR ERRORS
U 08B3, 088A,54 ;8378          JMP [LOOP.TB.3]                   ;ERROR LOOP IF ENABLED
                ;8379
U 08B4, 4043,15 ;8380          ADD LS[#2] TO WR[2]               ;ERROR 5 IN WR 2
                ;8381
U 08B5, E5F8,15 ;8382          CLR LS[OS]                        ;CLEAR INDEX
                ;8383  REPEAT.TB.5:
U 08B6, 36F6,15 ;8384          MOV LS[SHIFT.OS(4-0)] TO WR[0]    ;SET BITS 7-0 IN WR
U 08B7, 452C,15 ;8385          BIC LS[#FFFFFF00] TO WR[0]        ;CLEAR OTHER BITS IN WR
```

```
U 08B8, 4772,15 ;8386          BIS LS[ECC.DIS] TO WR[0]          ;SET ECC DISABLE BIT IN WR
U 08B9, 4774,15 ;8387          BIS LS[DIAG.CHK] TO WR[0]         ;ALSO SET DIAG CHK BIT
U 08BA, 8A17,FC ;8388          JSR [WRITE.CSR1]                  ;SET ECC DIS AND DIAG CHK BITS IN CSR 1
                ;8389                                            ; SET CKBT PATTERN IN BITS 6-0 (7 IS
                ;8390                                            ; ALSO SET BUT IS NOT USED IN CSR 1)
                ;8391   LOOP.TB.5:
U 08BB, E58A,15 ;8392          CLR LS[ERROR.MASK]                ;CHECK ALL BITS
U 08BC, 3E83,15 ;8393          MOV WR[2] TO LS[ERROR.NUMBER]     ;SET UP ERROR NUMBER IN LS (ERROR 5)
U 08BD, B6F6,95 ;8394          MOV LS[SHIFT.OS(4-0)] TO WR[1]    ;EXPECTED DATA FOR DATA BITS
U 08BE, 9D9C,F5 ;8395          MEM.REQ[MAINT.ECC.DATA] ADRS[ZERO] DT[LONG]
                ;8396                                            ;EXECUTE ECC CHIP DATA TEST
U 08BF, B2F6,15 ;8397          WRITE.MEM LS[SHIFT.OS(4-0)]       ;WRITE DATA OF SHIFTING 1'S IN ECC CHIP
                ;8398                                            ; CKBTS ALSO GET SHIFTING 1'S FROM CSR 1
U 08C0, 3022,15 ;8399          MOV MEM.DATA TO WR[0]             ;GET DATA BITS RESULT
U 08C1, 0869,3C ;8400          JSR [CHECK.RESULT]                ;CHECK FOR ERROR
U 08C2, 088B,B4 ;8401          JMP [LOOP.TB.5]                   ;ERROR LOOP IF ENABLED
                ;8402
U 08C3, 3E8B,95 ;8403          MOV WR[3] TO LS[ERROR.MASK]       ;STORE MASK IN LS
U 08C4, FF82,15 ;8404          INC LS[ERROR.NUMBER]              ;ERROR 6
U 08C5, B6F6,95 ;8405          MOV LS[SHIFT.OS(4-0)] TO WR[1]    ;EXPECTED DATA
U 08C6, 9D9D,75 ;8406          MEM.REQ[READ.CSR] ADRS[#0] DT[LONG]
                ;8407                                            ;READ CSR 0 FOR CKBTS
U 08C7, 3022,15 ;8408          MOV MEM.DATA TO WR[0]             ;GET RESULT OF CKBTS
U 08C8, 0869,3C ;8409          JSR [CHECK.RESULT]                ;CHECK FOR ERRORS
U 08C9, 088B,B4 ;8410          JMP [LOOP.TB.5]                   ;ERROR LOOP IF ENABLED
                ;8411
U 08CA, 7FF8,15 ;8412          INC LS[OS]                        ;INCREMENT INDEX FOR NEXT PATTERN
U 08CB, B6F8,15 ;8413          MOV LS[OS] TO WR[0]               ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                ;8414          BIT LS[BIT5] WITH WR[0],          ;SEE IF OS HAS INCREMENTED TO 20(H)
U 08CC, 594A,35 ;8415           DT(LONG)&SET.ALU.CC              ; AND SET CONDITION CODES
U 08CD, 088B,69 ;8416          JMP.IF[BITS.CLR] TO [REPEAT.TB.5] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                ;8417   END.TB:
```

```
;8418   .PAGE   "  TEST C ECC Checkbit Generation Test (M8391  MCT Module)"
;8419   ;++
;8420   ;
;8421   ; Test Description:
;8422   ;
;8423   ;       This test checks the checkbit generation logic in the ECC chips on
;8424   ;       the MCT module.  Data patterns are used which will produce known CKBTs
;8425   ;       which are written in CSR 0 to be checked. The patterns used will detect
;8426   ;       any stuck at or shorted signals on PART SYND xx L and BUS ARRAY CB x L
;8427   ;       lines.  It also tests the basic generate logic.  The data path follows:
;8428   ;       A MEM.REQ is issued with MF field=MAINT.ECC.DATA.  The LS address used
;8429   ;       contains data of 0(H) for branching later in the test.  This data is
;8430   ;       loaded into the VAR and a dispatch is made to the proper routine (see
;8431   ;       dispatch description at the beginning of this listing).  At the dis-
;8432   ;       patch address, memory busy and rotate disable are set, the PA bits are
;8433   ;       enabled and transceivers on the CPU module are enabled to allow data
;8434   ;       from the CPU to be received.  In the next cycle, the data rotators are
;8435   ;       clocked to disable rotation and the events of the previous cycle are
;8436   ;       repeated (except SET ADDR PH is not needed).  The following cycle opens
;8437   ;       the ECC latches, enables the output of the data rotators on the ARRAY
;8438   ;       BUS, clears any errors in the CSR, keeps memory busy set, and branches
;8439   ;       on L CPU DR.  CPU DR (data request) comes up when the CPU does a MOV u-
;8440   ;       instruction.  In this case, the CPU is sending the test data pattern
;8441   ;       for the rotator PALs.  If L CPU DR is not yet asserted, memory busy is
;8442   ;       dropped and the memory u-code will loop waiting for it.  Once L CPU DR
;8443   ;       is asserted, the next u-cycle will set 2nd memory cycle, open the ECC
;8444   ;       latches, enable the rotator PAL outputs onto the ARRAY BUS, and keep
;8445   ;       the CPU transceivers enabled.  Setting 2nd memory cycle keeps the data,
;8446   ;       sent over the MC BUS from the CPU, latched inside the rotator PAL (the
;8447   ;       data was loaded in the last u-cycle).  The next cycle keeps the ECC
;8448   ;       latches open, keeps the rotator PAL output on the ARRAY BUS, and waits
;8449   ;       for D DATA REC to be asserted. DATA RCVD is asserted by the CPU at the
;8450   ;       end (P2) of the CPU MOV instruction.  Looping here until the signal is
;8451   ;       asserted before asserting memory busy assures that the CPU will move on
;8452   ;       to the next u-instruction.  When D DATA REC is asserted, the memory u-
;8453   ;       code will branch on LVA00 (which was cleared at the beginning of the
;8454   ;       CPU.MEM.REQ instruction) to the proper test.  This u-cycle will assert
;8455   ;       memory busy, open the ECC latches, enable the rotator PAL outputs onto
;8456   ;       the ARRAY BUS, and enable the checkbits from CSR 1 onto the checkbit
;8457   ;       ARRAY BUS.  It will then branch to the proper test according to the
;8458   ;       state of ECC DIS and DIAG CHK in CSR 1.  CSR 1 was loaded prior to
;8459   ;       issueing the MEM.REQ instruction.  For this test, DIAG CHK is clear and
;8460   ;       ECC DIS is set.  Bits 0-6 contain the complement of the expected CKBTs
;8461   ;       which will be latched in the ECC chip before generating the new CKBTs.
;8462   ;       At the branch destination, generate is set, 2ND MEMORY CYCLE is cleared
;8463   ;       and memory busy is left asserted.  The following cycle repeats the last
;8464   ;       one except 2ND MEMORY CYCLE is already cleared.  The next cycle repeats
;8465   ;       the previous cycle and enables the generated CKBTs onto the CB BUS.
;8466   ;       The following cycle repeats the last cycle and also clocks the data on
;8467   ;       the CB bus into CSR 0.  The following cycles enable the ECC data onto
;8468   ;       the MC BUS through the data rotators, loops waiting for CPU DATA REQ
;8469   ;       and then CPU DATA RCVD, and returns to idle.  Data from the ECC chip is
;8470   ;       checked to assure that generate did not alter it, and CSR 0 is read to
;8471   ;       check that the correct CKBTs were generated.  Data patterns are as fol-
;8472   ;       lows (CKBTs are in binary starting at CT and going down to C1):  All
```

```
;8473    ;       0's which produces CKBTs of 0111100(B), all 1's which produces the same
;8474    ;       CKBTs as all 0's, data of 80000000(H) which produces CKBTs of 1000011
;8475    ;       (B), 1(H) which produces CKBTs of 1100100(B), 2(H) which produces CKBTs
;8476    ;       of 0100101(B),4(H) which produces CKBTs of 0100110, 10(H) which pro-
;8477    ;       duces CKBTs of 0100000, and 100(H) which produces CKBTs of 1010100(B).
;8478    ;       Please note that the CKBTs read back from the ECC chip are inverted
;8479    ;       before being written into CSR 0.  The data actually received will be
;8480    ;       the complement of the CKBT values shown above.
;8481    ;
;8482    ; Assumptions:
;8483    ;
;8484    ;       It is assumed that all previous tests have run successfully.
;8485    ;
;8486    ; Test Steps:
;8487    ;
;8488    ;       1)  Set up error mask, error number, and module number in LS (for
;8489    ;           error printouts) and clear previous error number in LS.
;8490    ;       2)  Write CSR 1 to clear DIAG CHK, set ECC DIS, and write complement of
;8491    ;           expected CKBTs in CSR 1 bits 6-0.
;8492    ;       3)  Load WR 1 with data pattern from LS (expected ECC data).
;8493    ;       4)  Issue a MEM.REQ with DT=longword (11) and MF=MAINT.ECC.DATA using
;8494    ;           a location in LS that contains a 0(H) as the address (this is load-
;8495    ;           ed into the VAR and used to branch later).
;8496    ;       5)  Execute a WRITE.MEM LS with current ECC data pattern.
;8497    ;       6)  Execute a MOVE MEM.DATA TO WR[0] and check that data was not alter-
;8498    ;           ed.
;8499    ;       7)  Change error mask to check bits 6-0, load WR 1 with expected CKBT
;8500    ;           result, read CSR 0, and check.
;8501    ;       8)  Repeat steps 2 through 7 with next data pattern (step 6 is executed
;8502    ;           with data patterns of all 1's and all 0's only).
;8503    ;
;8504    ; Errors:
;8505    ;
;8506    ;       NOTE: The expected and received data is ECC data when error mask is
;8507    ;       all zeros.  Expected and received data is CKBTS generated by ECC
;8508    ;       chip when error mask is FFFFFF80.
;8509    ;
;8510    ;       Error 1 - Generate CKBT function altered data latched in ECC chip.
;8511    ;                 Data was all 0's.
;8512    ;       Error 2 - ECC CKBT generation or control failure with ECC data of all
;8513    ;                 0's.
;8514    ;       Error 3 - Generate CKBT function altered data latched in ECC chip.
;8515    ;                 Data was all 1's.
;8516    ;       Error 4 - ECC CKBT generation or control failure with ECC data of all
;8517    ;                 1's.
;8518    ;       Error 5 - ECC CKBT generation or control failure with ECC data of
;8519    ;                 80000000(H).
;8520    ;       Error 6 - ECC CKBT generation failure with ECC data of 1(H).
;8521    ;       Error 7 - ECC CKBT generation failure with ECC data of 2(H).
;8522    ;       Error 8 - ECC CKBT generation failure with ECC data of 4(H).
;8523    ;       Error 9 - ECC CKBT generation failure with ECC data of 10(H).
;8524    ;       Error A - ECC CKBT generation failure with ECC data of 100(H).
;8525    ;
;8526    ; Debug:
;8527    ;
```

```
;8528    ;       Two methods of debug will be explained.  If the MCT module is in a
;8529    ;       test station then the MCT can be single stepped making debug easier
;8530    ;       in some cases.  Otherwise only the CPU can be single stepped and this
;8531    ;       method will be explained also.  When single stepping the memory con-
;8532    ;       troller will help in debug, this section will explicitly suggest it.
;8533    ;
;8534    ;       Error 1 - This failure indicates a problem with the ECC chip associated
;8535    ;       with the failing bit or the signal CORR DIS L (C29).  While looping on
;8536    ;       error, scope the signal CORR DIS L (C29) going into the ECC chip which
;8537    ;       is associated with the failing data bit.  This signal should never go
;8538    ;       low as this test is looping.  If the signal reamins high, the ECC chip
;8539    ;       is probably bad.
;8540    ;
;8541    ;       Error 2 - This failure indicates a problem with the ECC chip's CKBT
;8542    ;       generation or the control lines to the ECC chip.  If ECC single step is
;8543    ;       not available, loop on error will have to be used to scope the signals
;8544    ;       involved.  If MCT single step is available, first step the CPU to the
;8545    ;       WRITE.MEM instruction, enable MCT single step, step the CPU to the MOV
;8546    ;       MEM.DATA TO WR[0], and step the memory u-code to 4th GENERATE u-cycle
;8547    ;       (4 cycles after the branch on ECC.DIS and DIAG.CHK branch).  Check that
;8548    ;       the input GENERATE H (C27) is high at the low word ECC chip.  This sig-
;8549    ;       nal comes directly from the control store PROM.  If OK, check the sig-
;8550    ;       nal OUTPUT CB/SYN H (C28) on the high word ECC chip for a high.  Also
;8551    ;       check pin 26 (OUTCBBUS) and 33 (GEN) to assure that they are grounded
;8552    ;       and check that pins 24 (LATCB) and 23 (HI WORD) are tied high.  Pin 34
;8553    ;       on the low word ECC chip (OUTSYND) should also be checked to assure
;8554    ;       that it is tied high.  Finally check that CSR CB EN L (C43) which en-
;8555    ;       ables CSR 1 onto the CB BUS is high.  If all these inputs are OK, check
;8556    ;       the PART SYND xx L signals into ECC high word.  These should be all
;8557    ;       1's.  If not, the low word ECC chip is bad, or the line itself is bad.
;8558    ;       If these are OK, the high word ECC chip should be outputing the correct
;8559    ;       CKBTS on the CB BUS.  The correct CKBTS for this error are 1000011(B)
;8560    ;       from CB T L to CB 1 L respectively.  Otherwise the high word ECC chip
;8561    ;       is bad.
;8562    ;
;8563    ;       Error 3 - See error 1.
;8564    ;
;8565    ;       Error 4 - Same as error 2.  The data inputs are different, but all the
;8566    ;       control inputs and CKBT outputs are identical.
;8567    ;
;8568    ;       Error 5 - This pattern should complement the resulting CKBTS.  The PART
;8569    ;       SYND x L lines from low word are still all 1's but the CKBTs from high
;8570    ;       word onto the CB bus should be 0111100.  All control signals are ident-
;8571    ;       ical to error 2.
;8572    ;
;8573    ;       Error 6 - This error indicates either a problem with the ECC chip or
;8574    ;       the PART SYND x L lines between the two ECC chips.  Single step as in
;8575    ;       error 2 and check PART SYND x L for 0100111(B) from T L to 1 L respec-
;8576    ;       tively.  The high word outputs on the CB BUS should be 0011011(B) from
;8577    ;       CB T to CB 1 respectively.
;8578    ;
;8579    ;       Error 7 - This error indicates either a problem with the ECC chip or
;8580    ;       the PART SYND x L lines between the two ECC chips.  Single step as in
;8581    ;       error 2 and check PART SYND x L for 1100110(B) from T L to 1 L respec-
;8582    ;       tively.  The high word outputs on the CB BUS should be 1011010(B) from
```

```
                    ;8583   ;       CB T to CB 1 respectively.
                    ;8584   ;
                    ;8585   ;       Error 8 - This error indicates either a problem with the ECC chip or
                    ;8586   ;       the PART SYND x L lines between the two ECC chips.  Single step as in
                    ;8587   ;       error 2 and check PART SYND x L for 1100101(B) from T L to 1 L respec-
                    ;8588   ;       tively.  The high word outputs on the CB BUS should be 1011001(B) from
                    ;8589   ;       CB T to CB 1 respectively.
                    ;8590   ;
                    ;8591   ;       Error 9 - This error indicates either a problem with the ECC chip or
                    ;8592   ;       the PART SYND x L lines between the two ECC chips.  Single step as in
                    ;8593   ;       error 2 and check PART SYND x L for 1100011(B) from T L to 1 L respec-
                    ;8594   ;       tively.  The high word outputs on the CB BUS should be 1011111(B) from
                    ;8595   ;       CB T to CB 1 respectively.
                    ;8596   ;
                    ;8597   ;       Error A - This error indicates a problem with the ECC chip.  To deter-
                    ;8598   ;       mine which one, single step as in error 2 and check PART SYND x L for
                    ;8599   ;       0010111(B) from T L to 1 L respectively.  The high word outputs on the
                    ;8600   ;       CB BUS should be 1011001(B) from CB T to CB 1 respectively.
                    ;8601   ;
                    ;8602   ;--
                    ;8603
                    ;8604   T.C:
                    ;8605
U 08CE, B65E,15     ;8606           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;8607                                           ; STATUS WORD
U 08CF, 3E80,15     ;8608           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;8609                                           ; BIT 15 INDICATES START OF TEST TO
                    ;8610                                           ; CONSOLE
U 08D0, 10E0,15     ;8611           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;8612   WAIT.TC.0:
U 08D1, 088D,14     ;8613           JMP [WAIT.TC.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                    ;8614                                           ; RESPOND
                    ;8615
U 08D2, 0A1A,AC     ;8616           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;8617
U 08D3, B700,15     ;8618           MOV LS[CKBTS.0111100] TO WR[0]  ;COMPLEMENT OF EXPECTED CKBTS TO WRITE
                    ;8619                                           ; IN CSR 1
U 08D4, 4772,15     ;8620           BIS LS[ECC.DIS] TO WR[0]        ;SET ECC DISABLE IN CSR 1 FOR BRANCHING
U 08D5, 8A17,FC     ;8621           JSR [WRITE.CSR1]                ;WRITE CSR 1 WITH DATA IN WR 0
                    ;8622
U 08D6, B701,95     ;8623           MOV LS[CKBTS.0111100] TO WR[3]  ;STORE COMPLEMENT OF EXPECTED CKBTS IN
                    ;8624                                           ; WR 3
U 08D7, A0C7,95     ;8625           COM WR[3]                       ;NOW HAVE EXPECTED CKBTS IN WR[3] (CKBTS
                    ;8626                                           ; FROM ECC CHIP ARE COMPLEMENTED WHEN
                    ;8627                                           ; THEY ARE READ)
U 08D8, 3713,15     ;8628           MOV LS[#FFFFFF80] TO WR[2]      ;SET UP ERROR MASK FOR 7 BITS
                    ;8629   LOOP.TC.1:
U 08D9, B640,15     ;8630           MOV LS[#1] TO WR[0]             ;1 IN WR
U 08DA, BE82,15     ;8631           MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
U 08DB, B69C,95     ;8632           MOV LS[ZERO] TO WR[1]           ;EXPECTED DATA
U 08DC, E58A,15     ;8633           CLR LS[ERROR.MASK]              ;SET UP ERROR MASK TO CHECK ALL BITS
U 08DD, 9D9C,F5     ;8634           MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                    ;8635                                           ;SET UP TO WRITE ECC DATA
U 08DE, B29C,15     ;8636           WRITE.MEM LS[#0]                ;WRITE ECC DATA OF 0'S IN ECC CHIP
U 08DF, 3022,15     ;8637           MOV MEM.DATA TO WR[0]           ;CHECK THAT ECC DATA WAS NOT ALTERED
```

```
U 08E0, 0869,3C ;8638          JSR [CHECK.RESULT]                ;CHECK FOR ERRORS
U 08E1, 888D,94 ;8639          JMP [LOOP.TC.1]                   ;ERROR LOOP IF ENABLED
                ;8640
U 08E2, FF82,15 ;8641          INC LS[ERROR.NUMBER]              ;ERROR 2
U 08E3, 2006,95 ;8642          MOV WR[3] TO WR[1]                ;EXPECTED CKBT DATA
U 08E4, BE8B,15 ;8643          MOV WR[2] TO LS[ERROR.MASK]       ;CHANGE ERROR MASK TO CHECK 7 BITS (6-0)
U 08E5, 9D9D,75 ;8644          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                ;8645                                            ;READ GENERATED CKBTS FROM CSR0
U 08E6, 3022,15 ;8646          MOV MEM.DATA TO WR[0]             ;GET THE DATA
U 08E7, 0869,3C ;8647          JSR [CHECK.RESULT]                ;CHECK FOR ERRORS
U 08E8, 888D,94 ;8648          JMP [LOOP.TC.1]                   ;ERROR LOOP IF ENABLED
                ;8649
                ;8650    LOOP.TC.3:
U 08E9, 36C0,15 ;8651          MOV LS[#3(H)] TO WR[0]            ;3 IN WR
U 08EA, BE82,15 ;8652          MOV WR[0] TO LS[ERROR.NUMBER]     ;SET UP ERROR NUMBER IN LS
U 08EB, 369E,95 ;8653          MOV LS[ONES] TO WR[1]             ;EXPECTED DATA
U 08EC, E58A,15 ;8654          CLR LS[ERROR.MASK]                ;SET UP ERROR MASK TO CHECK ALL BITS
U 08ED, 9D9C,F5 ;8655          MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                ;8656                                            ;SET UP TO WRITE ECC DATA
U 08EE, 329E,15 ;8657          WRITE.MEM LS[ONES]                ;WRITE ECC DATA OF 1'S IN ECC CHIP
U 08EF, 3022,15 ;8658          MOV MEM.DATA TO WR[0]             ;CHECK THAT ECC DATA WAS NOT ALTERED
U 08F0, 0869,3C ;8659          JSR [CHECK.RESULT]                ;CHECK FOR ERRORS
U 08F1, 888E,94 ;8660          JMP [LOOP.TC.3]                   ;ERROR LOOP IF ENABLED
                ;8661
U 08F2, FF82,15 ;8662          INC LS[ERROR.NUMBER]              ;ERROR 4
U 08F3, 2006,95 ;8663          MOV WR[3] TO WR[1]                ;EXPECTED CKBT DATA
U 08F4, BE8B,15 ;8664          MOV WR[2] TO LS[ERROR.MASK]       ;CHANGE ERROR MASK TO CHECK 7 BITS (6-0)
U 08F5, 9D9D,75 ;8665          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                ;8666                                            ;READ GENERATED CKBTS FROM CSR0
U 08F6, 3022,15 ;8667          MOV MEM.DATA TO WR[0]             ;GET THE DATA
U 08F7, 0869,3C ;8668          JSR [CHECK.RESULT]                ;CHECK FOR ERRORS
U 08F8, 888E,94 ;8669          JMP [LOOP.TC.3]                   ;ERROR LOOP IF ENABLED
                ;8670
U 08F9, B67F,15 ;8671          MOV LS[BIT31] TO WR[2]            ;DATA OF 80000000 TO WRITE IN ARRAY
U 08FA, 3703,95 ;8672          MOV LS[CKBTS.1000011] TO WR[3]    ;COMPLEMENT OF EXPECTED CKBTS
U 08FB, 0890,CC ;8673          JSR [CHECK.TC.CKBT]               ;CHECK CKBT GENERATION  ERROR 5
                ;8674
U 08FC, 3641,15 ;8675          MOV LS[#1] TO WR[2]               ;DATA OF 1(H) TO WRITE IN ARRAY
U 08FD, 3705,95 ;8676          MOV LS[CKBTS.1100100] TO WR[3]    ;COMPLEMENT OF EXPECTED CKBTS
U 08FE, 0890,CC ;8677          JSR [CHECK.TC.CKBT]               ;CHECK CKBT GENERATION  ERROR 6
                ;8678
U 08FF, B643,15 ;8679          MOV LS[#2] TO WR[2]               ;DATA OF 2(H) TO WRITE IN ARRAY
U 0900, B707,95 ;8680          MOV LS[CKBTS.0100101] TO WR[3]    ;COMPLEMENT OF EXPECTED CKBTS
U 0901, 0890,CC ;8681          JSR [CHECK.TC.CKBT]               ;CHECK CKBT GENERATION  ERROR 7
                ;8682
U 0902, B645,15 ;8683          MOV LS[#4] TO WR[2]               ;DATA OF 4(H) TO WRITE IN ARRAY
U 0903, 3709,95 ;8684          MOV LS[CKBTS.0100110] TO WR[3]    ;COMPLEMENT OF EXPECTED CKBTS
U 0904, 0890,CC ;8685          JSR [CHECK.TC.CKBT]               ;CHECK CKBT GENERATION  ERROR 8
                ;8686
U 0905, B649,15 ;8687          MOV LS[#10] TO WR[2]              ;DATA OF 10(H) TO WRITE IN ARRAY
U 0906, B708,95 ;8688          MOV LS[CKBTS.0100000] TO WR[3]    ;COMPLEMENT OF EXPECTED CKBTS
U 0907, 0890,CC ;8689          JSR [CHECK.TC.CKBT]               ;CHECK CKBT GENERATION  ERROR 9
                ;8690
U 0908, B651,15 ;8691          MOV LS[#100] TO WR[2]             ;DATA OF 100(H) TO WRITE IN ARRAY
U 0909, B70D,95 ;8692          MOV LS[CKBTS.1010100] TO WR[3]    ;COMPLEMENT OF EXPECTED CKBTS
```

```
U 090A, 0890,CC ;8693            JSR [CHECK.TC.CKBT]                ;CHECK CKBT GENERATION  ERROR A
                ;8694
U 090B, 8891,84 ;8695            JMP [END.TC]                       ;DONE WITH TEST
                ;8696    ;
                ;8697    ; SUBROUTINE FOR TEST
                ;8698    ;
                ;8699
                ;8700    CHECK.TC.CKBT:
U 090C, FF82,15 ;8701            INC LS[ERROR.NUMBER]               ;INC ERROR NUMBER
U 090D, A0C7,95 ;8702            COM WR[3]                          ;NOW WR 3 CONTAINS EXPECTED CKBTS
U 090E, BE11,15 ;8703            MOV WR[2] TO LS[T8]                ;STORE DATA TO WRITE IN ARRAY IN LS
                ;8704    LOOP.TC.5.A:
U 090F, 2006,95 ;8705            MOV WR[3] TO WR[1]                 ;EXPECTED CKBT DATA
U 0910, 9D9C,F5 ;8706            MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                ;8707                                               ;SET UP TO WRITE ECC DATA
U 0911, 3210,15 ;8708            WRITE.MEM LS[T8]                   ;WRITE ECC DATA IN ECC CHIP
U 0912, 3022,15 ;8709            MOV MEM.DATA TO WR[0]              ;COMPLETE U-CYCLE BUT DO NOT CHECK RESULT
U 0913, 9D9D,75 ;8710            MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                ;8711                                               ;READ GENERATED CKBTS FROM CSR0
U 0914, 3022,15 ;8712            MOV MEM.DATA TO WR[0]              ;GET THE DATA
U 0915, 0869,3C ;8713            JSR [CHECK.RESULT]                 ;CHECK FOR ERRORS
U 0916, 8890,F4 ;8714            JMP [LOOP.TC.5.A]                  ;ERROR LOOP FOR ERRORS 5 THRU A IF
                ;8715                                               ; ENABLED
U 0917, 5B00,14 ;8716            RETURN                             ;DONE
                ;8717
                ;8718    END.TC:
```

```
;8719    .PAGE   "  TEST D ECC Error (no correction) Test (M8391  MCT Module  M8390 CPU Module)"
;8720    ;++
;8721    ;
;8722    ; Test Description:
;8723    ;
;8724    ;       This test checks that the ECC chip will correctly detect a single or
;8725    ;       double bit error and that RDS, CRD, and ERR SUM will work correctly.
;8726    ;       The skip on error sum logic on the CPU module is also checked.  The INH
;8727    ;       REP CRD bit in CSR 1 is toggled on a CRD error to assure that it in-
;8728    ;       hibits CRD from setting in CSR 1.  The data path is identical to the
;8729    ;       ARY BUS and ECC latches test preceeding the CKBT generation test.  See
;8730    ;       that test for a data path description.  Data patterns used are all 0's
;8731    ;       as ECC data with the following CKBT data patterns (from CB T to CB 1
;8732    ;       respectively):  0111100(B) which produces no error,  1100100 which
;8733    ;       should produce a single bit (CRD) error, and 1111101(B) which should
;8734    ;       produce a double bit (RDS) error.  Note that CKBTs are written and read
;8735    ;       as complemented data due to the hardware.
;8736    ;
;8737    ; Assumptions:
;8738    ;
;8739    ;       It is assumed that all previous tests have run successfully.  The skip
;8740    ;       on ERR SUM logic of the CPU module is tested here, so the CPU module
;8741    ;       may cause a failure.
;8742    ;
;8743    ; Test Steps:
;8744    ;
;8745    ;       1)  Set up error mask, error number, and module number in LS (for
;8746    ;           error printouts) and clear previous error number in LS.
;8747    ;       2)  Write CSR 1 with data to set both DIAG CHK and ECC DIS.  Also load
;8748    ;           CKBT portion (bits 0-6) with CKBTs to produce no error (0111100(B)
;8749    ;           from CB T to CB 1 respectively).  CKBT data must be inverted before
;8750    ;           writing into CSR.
;8751    ;       3)  Clear WR[1].  This is expected data from ECC chip.
;8752    ;       4)  Issue a MEM.REQ with DT=longword (11) and MF=MAINT.ECC.DATA using
;8753    ;           a location in LS that contains a 0(H) as the address (this is load-
;8754    ;           ed into the VAR and used to branch later).
;8755    ;       5)  Execute a WRITE.MEM LS with data of all 0's.
;8756    ;       6)  Execute a MOV MEM.DATA to WR[0].  Check result for all 0's (no
;8757    ;           change).
;8758    ;       7)  Change error mask to check bits 14-23 and 25-31 of CSR 1.  Load WR
;8759    ;           1 with expected CSR 1 data.
;8760    ;       8)  Read CSR 1 and check for DIAG CHK and ECC DIS set, others clear.
;8761    ;       9)  Change module code to include CPU module in callout. Execute a skip
;8762    ;           on ERR SUM NOT and check for a skip.
;8763    ;       10) Write CSR 1 with CKBT data to produce a single bit error with data
;8764    ;           of all 0's (CKBTs = 1100100(B) from CB T to CB 1 respectively.  The
;8765    ;           CKBTs must be inverted before writing into CSR).
;8766    ;       11) Restore data error mask to 32 bits and restore module callout to
;8767    ;           MCT module only.
;8768    ;       12) Clear WR[1].  This is expected data from ECC chip.  Issue a MEM.REQ
;8769    ;           as in step 4 above.
;8770    ;       13) Execute a WRITE.MEM LS with data pattern of 0's.
;8771    ;       14) Execute a MOV MEM.DATA to WR[0].  Check result for all 0's (no
;8772    ;           change).
;8773    ;       15) Change error mask to check bits 14-23 and 25-31 of CSR 1.  Load WR
```

```
;8774    ;           1 with expected CSR 1 data.
;8775    ;       16) Read CSR 1 and check for CRD (bit 30), DIAG CHK and ECC DIS set,
;8776    ;           other bits clear.
;8777    ;       17) Change module code to include CPU module in callout.  Execute a
;8778    ;           skip on ERR SUM NOT and check that no skip occurs.  Clear CSR 2 and
;8779    ;           check that CRD bit in CSR 1 is cleared.
;8780    ;       18) Repeat steps 10 through 17 with CKBITs to produce a double error.
;8781    ;           Check for RDS (bit 31), DIAG CHK, and ECC DIS set in CSR 1, other
;8782    ;           bits clear.  CKBITs = 1111101(B) from CB 7 to CB 1 respectively.
;8783    ;           CKBITs must be inverted before writing in CSR.
;8784    ;       19) Repeat steps 10 through 17 with CKBITs for a single error again but
;8785    ;           set INH REP CRD in CSR 1.  Check that no skip occurs but that the
;8786    ;           CRD bit in CSR 1 does not set.
;8787    ;       20) Check that error sum reaches CPU quickly enough.
;8788    ;
;8789    ; Errors:
;8790    ;
;8791    ;       Error 1 - ECC data was modified using data of 0 and CKBITs for no error.
;8792    ;                 Correction was also disabled.
;8793    ;       Error 2 - CSR 1 error bits were not clear even though data and CKBITs
;8794    ;                 should have produced no error.
;8795    ;       Error 3 - Skip on error sum not in CPU failed to skip on no error.
;8796    ;       Error 4 - ECC data was modified using data of 0 and CKBITs for a single
;8797    ;                 error even though correction was disabled.
;8798    ;       Error 5 - CSR 1 error bits not set up correctly on a single bit (CRD)
;8799    ;                 error.
;8800    ;       Error 6 - Skip on error sum not in CPU skipped on single bit error.
;8801    ;       Error 7 - Write to CSR 2 did not clear CRD bit in CSR 1.
;8802    ;       Error 8 - ECC data was modified using data of 0 and CKBITs for a double
;8803    ;                 bit error even though correction was disabled.
;8804    ;       Error 9 - CSR 1 error bits not set up correctly on a double bit (RDS)
;8805    ;                 error.
;8806    ;       Error A - Skip on error sum not in CPU skipped on double bit error.
;8807    ;       Error B - Write to CSR 2 did not clear RDS bit in CSR 2.
;8808    ;       Error C - CSR 1 error bits not all clear on execution of a single bit
;8809    ;                 (CRD) error with INH REP CRD set.
;8810    ;       Error D - Skip on error sum not in CPU skipped on single bit error with
;8811    ;                 INH REP CRD set (DIS ECC was asserted so error sum should
;8812    ;                 still have been asserted).
;8813    ;       Error E - Skip on error sum not skipped even though error sum was
;8814    ;                 asserted on MCT.
;8815    ;
;8816    ; Debug:
;8817    ;
;8818    ;       Two methods of debug will be explained.  If the MCT module is in a
;8819    ;       test station then the MCT can be single stepped making debug easier
;8820    ;       in some cases.  Otherwise only the CPU can be single stepped and this
;8821    ;       method will be explained also.  When single stepping the memory con-
;8822    ;       troller will help in debug, this section will explicitly suggest it.
;8823    ;
;8824    ;       Error 1 - This error indicates a basic problem with the ECC chip.
;8825    ;       Suspect the ECC chip associated with the failing bit in the error re-
;8826    ;       port.
;8827    ;
;8828    ;       Error 2 - This error indicates that either the CSR error bits are not
```

```
;8829   ;       getting cleared, the ECC chip is producing an erroneous error signal,
;8830   ;       or that the CSR 1 logic is faulty.  With halt on error set, the error
;8831   ;       signals from the ECC chip can be checked after the test halts on error.
;8832   ;       Check the signals ERROR L and SINGLE ERROR L coming from the high word
;8833   ;       ECC chip.  Both should be high.  If not, one of the ECC chips is prob-
;8834   ;       ably faulty.  If these outputs are OK, start over at the memory idle
;8835   ;       loop, enable MCT single step (if available), step the CPU to the
;8836   ;       MEM.REQ, and step the MCT u-code to the second cycle after the dispatch
;8837   ;       address (CLR ERR cycle).  Check the signal CLR ERR L (C40) at the CSR 1
;8838   ;       PALs for a low and CPU ERR SUM CLK H (C48) for a high.  Step to the next
;8839   ;       cycle and check that CPU ERR SUM CLK H goes low.  If it does, the PAL should
;8840   ;       have been cleared.  If any of these outputs were high (CSR bits 14-23)
;8841   ;       when the test failed, the PAL is probably faulty if the inputs are
;8842   ;       OK (the outputs are only enabled when a read CSR is performed, so they
;8843   ;       can not be looked at directly here).  If MCT single step is not avail-
;8844   ;       able, then loop on error and check.  If bit 30 or 31 is failing, check
;8845   ;       the output signals L RDS H and L CRD H at the CSR CONTROL PAL for lows
;8846   ;       after the CLR ERR memory u-cycle.  The inputs during this cycle should
;8847   ;       be a high on CPH/UBL and on CSR ERR SUM CLK H (C42).  CPH/UBL comes
;8848   ;       from the CONTROL PREFETCH PAL and has already been tested high in a
;8849   ;       previous test.  If these signals are OK, then the PAL is bad if the
;8850   ;       outputs are not low.  The last possibility if bit 30 or 31 is failing
;8851   ;       is that the buffer or connections to it are bad that read back CRD and
;8852   ;       RDS onto the MC BUS.  This can be checked after halting on error.
;8853   ;
;8854   ;       Error 3 - This error indicates a problem with the ERR SUM logic on the
;8855   ;       MCT module or the skip on ERR SUM NOT on the CPU module.  After halting
;8856   ;       on error, check the signal ERR SUM H on the MCT module. If this is low,
;8857   ;       the problem is probably on the CPU module or the backplane connection
;8858   ;       to the CPU.  On the CPU module check the connection ERR SUM H into the
;8859   ;       SEQ.CTL PAL for a low.  If this is OK, the SEQ.CTL PAL is probably bad,
;8860   ;       but the inputs CSR 04 H through CSR 00 H can be checked for a 1D(H) and
;8861   ;       JUMP INST L can be checked for a high to make sure.  If ERR SUM H on
;8862   ;       the MCT module is high, then trace it back to a PAL.  If either CSR 1A
;8863   ;       or CSR 1B is causing the problem, that PAL is probably bad.  If the
;8864   ;       signal DATA ERR L is low, then check the CSR CONTROL PAL output.  If
;8865   ;       the output is low here, the CSR CONTROL PAL is probably bad.
;8866   ;
;8867   ;       Error 4 - The ECC chip itself is the most likely suspect.  Check the
;8868   ;       signal COR DIS L going to both chips for a low to be sure.
;8869   ;
;8870   ;       Error 5 - This error could be caused by the ECC chip, faulty branch
;8871   ;       logic, or the CSR 1 logic.  It is unlikely that bits 14-23 would fail
;8872   ;       if this is the first test.  If bit 30 or 31 is failing, first check the
;8873   ;       signals ERROR L and SINGLE ERR L out of the high word ECC chip.  Both
;8874   ;       of these signals should be low.  If not, one of the ECC chips is prob-
;8875   ;       ably bad.  If these signals are OK, check them at the input to the CSR
;8876   ;       CONTROL PAL.  If OK there, and MCT single step is available, step the
;8877   ;       CPU to the WRITE.MEM LS instruction.  The memory u-code will be looping
;8878   ;       waiting for DATA RCVD from the CPU.  Enable MCT single step, then step
;8879   ;       the CPU once more to the MOV MEM.DATA TO WR[0] instruction.  Now step
;8880   ;       the memory u-code 4 cycles to the branch on error cycle (BEN0=ERROR.L).
;8881   ;       Check the signal ERROR L into the BEN 0 MUX for a low.  Also check the
;8882   ;       select lines for 011(B) on SEL 4, 2, and 1 respectively.  If these are
;8883   ;       OK, step the memory one more cycle and it should be on the u-cycle that
```

```
;8884  ;       asserts CSR ERR ADDR CLK A L (C45).  If the branch failed, the BEN 0
;8885  ;       MUX is bad.  Otherwise check the inputs to the CSR CONTROL PAL for the
;8886  ;       following:  CSR ERR ADDR CLK A L, CSR ERR SUM CLK H, SINGLE ERR L, and
;8887  ;       ERROR L should all be low, CPH/UBL should be high, INH REP CRD H should
;8888  ;       be low, and WR CSR L should be high.  If these inputs are correct, then
;8889  ;       L RDS H should be low and L CRD H should be high.  The PAL is bad if
;8890  ;       these outputs are incorrect.  If OK here, check that the signals get to
;8891  ;       the QUAD BUFFER chip.  The buffer is probably bad if everything is OK
;8892  ;       up to its inputs.  One other possibility is the L CRD H and L RDS H are
;8893  ;       changing later in the u-flow.  Unless the CSR CONTROL PAL is bad, this
;8894  ;       should not occur as long as a WR CSR or CSR ERR SUM CLK is not assert-
;8895  ;       ed.  If MCT single step is not available, loop on error and sync on CSR
;8896  ;       ERR ADDR CLK A L to check the inputs to the CSR CONTROL PAL.  If CSR
;8897  ;       ERR ADDR CLK A L does not get asserted, the branch logic may be faulty.
;8898  ;
;8899  ;       Error 6 - This error indicates a problem with the ERR SUM logic on the
;8900  ;       MCT module, or the branch on ERR SUM logic on the CPU module.  After
;8901  ;       halting on error, check the signal ERR SUM H on the MCT module for a
;8902  ;       high.  If this is OK, the problem is on the CPU board or the backplane
;8903  ;       connection to the CPU module.  Check the signal ERR SUM H at the input
;8904  ;       to the SEQ.CTL PAL for a high.  If this is OK, the SEQ.CTL PAL is prob-
;8905  ;       ably bad, but the inputs can be checked to make sure by stepping the
;8906  ;       CPU u-code to the skip on MEM.REF.OK instruction.  The input CSR 04 to
;8907  ;       CSR 00 should equal 1D(H) and JUMP INST L should be high.
;8908  ;         If ERR SUM H on the MCT module is low, check the signal DATA ERR L
;8909  ;       or one leg of the negated input 'OR' gate.  This should be low here
;8910  ;       and at the CSR CONTROL PAL output.  If the output at the PAL is wrong,
;8911  ;       check the inputs to the CSR CONTROL PAL as described in error 5 above.
;8912  ;
;8913  ;       Error 7 - This error indicates a problem with the CSR DATA ERROR
;8914  ;       CONTROL PAL or the inputs WR CSR L ;(C46) and CSR ERR SUM CLK H.  Check
;8915  ;       these inputs during the JSR [CLEAR.CSR2] intruction for a low on WR CSR
;8916  ;       L and a high on CSR ERR SUM CLK.  If these are OK, the PAL is probably
;8917  ;       bad.
;8918  ;
;8919  ;       Error 8 - Same as error 4.
;8920  ;
;8921  ;       Error 9 - This error indicates a problem with the ECC chips or the CSR
;8922  ;       CONTROL PAL logic.  After halting on error, check the output of the
;8923  ;       high word ECC chip for a low on ERROR L and a high on SINGLE ERR L.  If
;8924  ;       these are incorrect, the problem is probably with one of the ECC chips.
;8925  ;       Otherwise check the inputs at the CSR CONTROL PAL.  The outputs can be
;8926  ;       checked also to see if the problem is at the BUFFER chip.  If the out-
;8927  ;       puts are incorrect, check the other inputs to the CSR CONTROL PAL by
;8928  ;       single stepping or looping as described in error 5 above.  The only
;8929  ;       differance is that SINGLE ERROR L should be high, resulting in L CRD H
;8930  ;       being low and L RDS H being high.
;8931  ;
;8932  ;       Error A - This error probably indicates a problem with the CSR CONTROL
;8933  ;       PAL itself.
;8934  ;
;8935  ;       Error B - Same as error 7.
;8936  ;
;8937  ;       Error C - This error indicates a problem with the CSR CONTROL PAL or
;8938  ;       the input INH REP CRD H.  Check this input after halting on error for
```

```
                    ;8939   ;       a high.  If OK, the CSR CONTROL PAL is probably faulty.
                    ;8940   ;
                    ;8941   ;       Error D - This error probably indicates that the CSR CONTROL PAL is
                    ;8942   ;       faulty.  The signal L ECC DIS H can be checked after the halt on error
                    ;8943   ;       to make sure.  L ECC DIS H should be high.  The other inputs have been
                    ;8944   ;       checked in other errors.
                    ;8945   ;
                    ;8946   ;       Error E - This error indicates that ERROR SUM L is not reaching the
                    ;8947   ;       CPU quickly enough.  The stall before ERROR SUM is checked will last
                    ;8948   ;       for 2 memory cycles (until MEM.BUSY is dropped).  The ERROR SUM should
                    ;8949   ;       be able to propagate through the necessary logic in that time frame.
                    ;8950   ;       Suspect a slow chip if this error occurs.
                    ;8951   ;--
                    ;8952
                    ;8953   T.D:
                    ;8954
J 0918, B65E,15     ;8955           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;8956                                           ; STATUS WORD
U 0919, 3E80,15     ;8957           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;8958                                           ; BIT 15 INDICATES START OF TEST TO
                    ;8959                                           ; CONSOLE
U 091A, 10E0,15     ;8960           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;8961   WAIT.TD.0:
U 091B, 8891,B4     ;8962           JMP [WAIT.TD.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                    ;8963                                           ; RESPOND
                    ;8964
U 091C, 0A1A,AC     ;8965           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;8966
U 091D, 3673,95.    ;8967           MOV LS[ECC.DIS] TO WR[3]         ;SET ECC DISABLE IN WR 3
U 091E, 4775,95     ;8968           BIS LS[DIAG.CHK] TO WR[3]        ;ALSO SET DIAG CHK IN WR 3. NOW WR 3 IS
                    ;8969                                           ; EXPECTED DATA FOR ERROR 2
U 091F, B700,15     ;8970           MOV LS[CKBTS.0111100] TO WR[0]  ;PUT CKBTS FOR NO ERROR IN WR 0
U 0920, A0C0,15     ;8971           COM WR[0]                       ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 0921, 452C,15     ;8972           BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE
U 0922, 2EC6,15     ;8973           BIS WR[3] TO WR[0]              ;SET DIAG CHK AND ECC DIS IN WR
U 0923, 8A17,FC     ;8974           JSR [WRITE.CSR1]                ;WRITE CSR 1 WITH DATA FROM WR[0]
                    ;8975
U 0924, B629,15     ;8976           MOV LS[#FFFF] TO WR[2]           ;ERROR MASK FOR ERROR 2.  MASK LOW 16
                    ;8977                                           ; BITS (DO NOT CHECK)
U 0925, 455D,15     ;8978           BIC LS[VALID.ERR] TO WR[2]       ;CHECK VALID ERROR (BIT 14)
U 0926, C55F,15     ;8979           BIC LS[TB.PAR.ERR] TO WR[2]      ;AND CHECK TB PARITY ERROR (BIT 15)
U 0927, 4771,15     ;8980           BIS LS[BIT24] TO WR[2]           ;DON'T CHECK UNUSED BIT (BIT 24)
                    ;8981
                    ;8982   LOOP.TD.1:
U 0928, B640,15     ;8983           MOV LS[#1] TO WR[0]              ;1 IN WR
U 0929, BE82,15     ;8984           MOV WR[0] TO LS[ERROR.NUMBER]    ;SET UP ERROR NUMBER IN LS
U 092A, E58A,15     ;8985           CLR LS[ERROR.MASK]               ;CHECK ALL BITS FOR ERROR 1
U 092B, 2F82,95     ;8986           CLR WR[1]                        ;EXPECTED ECC DATA IS 0
U 092C, 9D9C,F5     ;8987           MEM.REG[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                    ;8988                                           ;SET UP TO WRITE CKBTS FROM CSR 1 AND
                    ;8989                                           ; DATA FROM LS
U 092D, B29C,15     ;8990           WRITE.MEM LS[ZERO]               ;WRITE ECC DATA OF 0
U 092E, 3022,15     ;8991           MOV MEM.DATA TO WR[0]            ;GET ECC DATA RESULT (SHOULD BE 0)
U 092F, 0869,3C     ;8992           JSR [CHECK.RESULT]               ;CHECK THAT ECC DATA WAS NOT MODIFIED
U 0930, 8892,84     ;8993           JMP [LOOP.TD.1]                  ;ERROR LOOP IF ENABLED
```

```
                    ;8994
U 0931, FF82,15     ;8995           INC LS[ERROR.NUMBER]                ;ERROR 2
U 0932, BE8B,15     ;8996           MOV WR[2] TO LS[ERROR.MASK]         ;CHANGE ERROR MASK TO CHECK BITS 14-23
                    ;8997                                               ; AND 25-31
U 0933, 2006,95     ;8998           MOV WR[3] TO WR[1]                  ;EXPECTED DATA (ECC DIS AND DIAG CHK SET)
U 0934, 9D45,75     ;8999           MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;9000                                               ;SET UP TO READ CSR 1
U 0935, 3022,15     ;9001           MOV MEM.DATA TO WR[0]                ;GET CONTENTS OF CSR 1
U 0936, 0869,3C     ;9002           JSR [CHECK.RESULT]                  ;CHECK RESULT FOR ERRORS
U 0937, 8892,84     ;9003           JMP [LOOP.TD.1]                     ;ERROR LOOP IF ENABLED
                    ;9004
U 0938, FF82,15     ;9005           INC LS[ERROR.NUMBER]                ;ERROR 3
                    ;9006   LOOP.TD.3:
U 0939, 5BC0,1D     ;9007           SKIP.IF[MEM.REF.OK]                 ;CHECK SKIP LOGIC.  THIS INSTRUCTION
                    ;9008                                               ; SHOULD SKIP THE JSR FOLLOWING
U 093A, 889A,2C     ;9009           JSR [ERROR.TD.3]                    ;REPORT SKIP ON ERROR SUM FAILURE
                    ;9010
U 093B, 3673,95     ;9011           MOV LS[ECC.DIS] TO WR[3]             ;SET ECC DISABLE IN WR 3
U 093C, 4775,95     ;9012           BIS LS[DIAG.CHK] TO WR[3]            ;ALSO SET DIAG CHK IN WR 3
U 093D, 3704,15     ;9013           MOV LS[CKBTS.1100100] TO WR[0]       ;PUT CKBTS FOR SINGLE BIT ERROR IN WR 0
U 093E, A0C0,15     ;9014           COM WR[0]                           ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 093F, 452C,15     ;9015           BIC LS[#FFFFFF00] TO WR[0]          ;CLEAR ALL BUT LOW BYTE
U 0940, 2EC6,15     ;9016           BIS WR[3] TO WR[0]                  ;SET DIAG CHK AND ECC DIS IN WR
U 0941, 8A17,FC     ;9017           JSR [WRITE.CSR1]                    ;WRITE CSR 1 WITH DATA FROM WR[0]
                    ;9018
U 0942, C77D,95     ;9019           BIS LS[CRD] TO WR[3]                 ;NOW WR 3 IS EXPECTED DATA FOR ERROR 5
                    ;9020
                    ;9021   LOOP.TD.4:
U 0943, 3644,15     ;9022           MOV LS[#4] TO WR[0]                  ;4 IN WR
U 0944, BE82,15     ;9023           MOV WR[0] TO LS[ERROR.NUMBER]        ;SET UP ERROR NUMBER IN LS
U 0945, E58A,15     ;9024           CLR LS[ERROR.MASK]                  ;CHECK ALL BITS FOR ERROR 4
U 0946, 2F82,95     ;9025           CLR WR[1]                           ;EXPECTED ECC DATA IS 0
U 0947, 9D9C,F5     ;9026           MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                    ;9027                                               ;SET UP TO WRITE CKBTS FROM CSR 1 AND
                    ;9028                                               ; DATA FROM LS
U 0948, B29C,15     ;9029           WRITE.MEM LS[ZERO]                  ;WRITE ECC DATA OF 0
U 0949, 3022,15     ;9030           MOV MEM.DATA TO WR[0]                ;GET ECC DATA RESULT (SHOULD BE 0)
U 094A, 0869,3C     ;9031           JSR [CHECK.RESULT]                  ;CHECK THAT ECC DATA WAS NOT MODIFIED
U 094B, 0894,34     ;9032           JMP [LOOP.TD.4]                     ;ERROR LOOP IF ENABLED
                    ;9033
U 094C, FF82,15     ;9034           INC LS[ERROR.NUMBER]                ;ERROR 5
U 094D, BE8B,15     ;9035           MOV WR[2] TO LS[ERROR.MASK]         ;CHANGE ERROR MASK TO CHECK BITS 14-23
                    ;9036                                               ; AND 25-31
U 094E, 2006,95     ;9037           MOV WR[3] TO WR[1]                  ;EXPECTED DATA (CRD, ECC DIS AND DIAG
                    ;9038                                               ; CHK SET)
U 094F, 9D45,75     ;9039           MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;9040                                               ;SET UP TO READ CSR 1
U 0950, 3022,15     ;9041           MOV MEM.DATA TO WR[0]                ;GET CONTENTS OF CSR 1
U 0951, 0869,3C     ;9042           JSR [CHECK.RESULT]                  ;CHECK RESULT FOR ERRORS
U 0952, 0894,34     ;9043           JMP [LOOP.TD.4]                     ;ERROR LOOP IF ENABLED
                    ;9044
U 0953, FF82,15     ;9045           INC LS[ERROR.NUMBER]                ;ERROR 6
                    ;9046   LOOP.TD.6:
U 0954, 5B00,1D     ;9047           SKIP.IF[MEM.REF.OK]                 ;CHECK SKIP LOGIC.  THIS INSTRUCTION
                    ;9048                                               ; SHOULD NOT SKIP
```

```
U 0955, 0895,74 ;9049           JMP [TEST.TD.7]                 ;NO ERROR IF HERE, CONTINUE TESTING
U 0956, 089A,5C ;9050           JSR [ERROR.TD.6]                ;REPORT SKIP ON ERROR SUM FAILURE IF HERE
                ;9051
                ;9052   TEST.TD.7:
U 0957, FF82,15 ;9053           INC LS[ERROR.NUMBER]            ;ERROR 7
U 0958, 457D,95 ;9054           BIC LS[CRD] TO WR[3]            ;EXPECTED DATA HAS CRD BIT CLEARED
                ;9055   LOOP.TD.7:
U 0959, 8A1E,3C ;9056           JSR [CLEAR.CSR2]                ;DO WRITE TO CSR 2
U 095A, 2006,95 ;9057           MOV WR[3] TO WR[1]              ;EXPECTED DATA (ECC DIS AND DIAG
                ;9058                                           ; CHK SET)
U 095B, 9D45,75 ;9059           MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;9060                                           ;SET UP TO READ CSR 1
U 095C, 3022,15 ;9061           MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1.  CRD BIT SHOULD
                ;9062                                           ; HAVE BEEN CLEARED BY WRITE TO CSR 2
U 095D, 0869,3C ;9063           JSR [CHECK.RESULT]              ;CHECK RESULT FOR ERRORS
U 095E, 8895,94 ;9064           JMP [LOOP.TD.7]                 ;ERROR LOOP IF ENABLED
                ;9065
                ;9066   TEST.TD.8:
U 095F, FF82,15 ;9067           INC LS[ERROR.NUMBER]            ;ERROR 8
U 0960, 3682,15 ;9068           MOV LS[ERROR.NUMBER] TO WR[0]   ;GET UPDATED ERROR NUMBER
U 0961, 3E10,15 ;9069           MOV WR[0] TO LS[T8]             ;AND STORE IN TEMP LS LOCATION
                ;9070
U 0962, 3673,95 ;9071           MOV LS[ECC.DIS] TO WR[3]        ;SET ECC DISABLE IN WR 3
U 0963, 4775,95 ;9072           BIS LS[DIAG.CHK] TO WR[3]       ;ALSO SET DIAG CHK IN WR 3.
U 0964, 370E,15 ;9073           MOV LS[CKBTS.1111101] TO WR[0]  ;PUT CKBTS FOR AN UNCORRECTABLE ERROR
                ;9074                                           ; IN WR 0
U 0965, A0C0,15 ;9075           COM WR[0]                       ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 0966, 452C,15 ;9076           BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE
U 0967, 2EC6,15 ;9077           BIS WR[3] TO WR[0]              ;SET DIAG CHK AND ECC DIS IN WR
U 0968, 8A17,FC ;9078           JSR [WRITE.CSR1]                ;WRITE CSR 1 WITH DATA FROM WR[0]
                ;9079
U 0969, 477F,95 ;9080           BIS LS[RDS] TO WR[3]            ;NOW WR 3 IS EXPECTED DATA FOR ERROR 8
                ;9081   LOOP.TD.8:
U 096A, B610,15 ;9082           MOV LS[T8] TO WR[0]             ;8 IN WR
U 096B, BE82,15 ;9083           MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
U 096C, E58A,15 ;9084           CLR LS[ERROR.MASK]              ;CHECK ALL BITS FOR ERROR 8
U 096D, 2F82,95 ;9085           CLR WR[1]                       ;EXPECTED ECC DATA IS 0
U 096E, 9D9C,F5 ;9086           MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                ;9087                                           ;SET UP TO WRITE CKBTS FROM CSR 1 AND
                ;9088                                           ; DATA FROM LS
U 096F, B29C,15 ;9089           WRITE.MEM LS[ZERO]              ;WRITE ECC DATA OF 0
U 0970, 3022,15 ;9090           MOV MEM.DATA TO WR[0]           ;GET ECC DATA RESULT (SHOULD BE 0)
U 0971, 0869,3C ;9091           JSR [CHECK.RESULT]              ;CHECK THAT ECC DATA WAS NOT MODIFIED
U 0972, 8896,A4 ;9092           JMP [LOOP.TD.8]                 ;ERROR LOOP IF ENABLED
                ;9093
U 0973, FF82,15 ;9094           INC LS[ERROR.NUMBER]            ;ERROR 9
U 0974, BE8B,15 ;9095           MOV WR[2] TO LS[ERROR.MASK]     ;CHANGE ERROR MASK TO CHECK BITS 14-23
                ;9096                                           ; AND 25-31
U 0975, 2006,95 ;9097           MOV WR[3] TO WR[1]              ;EXPECTED DATA (RDS, ECC DIS AND DIAG
                ;9098                                           ; CHK SET)
U 0976, 9D45,75 ;9099           MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;9100                                           ;SET UP TO READ CSR 1
U 0977, 3022,15 ;9101           MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 0978, 0869,3C ;9102           JSR [CHECK.RESULT]              ;CHECK RESULT FOR ERRORS
U 0979, 8896,A4 ;9103           JMP [LOOP.TD.8]                 ;ERROR LOOP IF ENABLED
```

```
                    ;9104
U 097A, FF82,15     ;9105          INC LS[ERROR.NUMBER]              ;ERROR A
                    ;9106    LOOP.TD.A:
U 097B, 5B00,1D     ;9107          SKIP.IF[MEM.REF.OK]               ;CHECK SKIP LOGIC.  THIS INSTRUCTION
                    ;9108                                            ; SHOULD NOT SKIP
U 097C, 8897,E4     ;9109          JMP [TEST.TD.B]                   ;NO ERROR IF HERE, CONTINUE TESTING
U 097D, 889A,8C     ;9110          JSR [ERROR.TD.A]                  ;REPORT SKIP ON ERROR SUM FAILURE IF HERE
                    ;9111
                    ;9112    TEST.TD.B:
U 097E, FF82,15     ;9113          INC LS[ERROR.NUMBER]              ;ERROR B
U 097F, C57F,95     ;9114          BIC LS[RDS] TO WR[3]              ;EXPECTED DATA HAS RDS BIT CLEARED
                    ;9115    LOOP.TD.B:
U 0980, 8A1E,3C     ;9116          JSR [CLEAR.CSR2]                  ;DO WRITE TO CSR 2
U 0981, 2006,95     ;9117          MOV WR[3] TO WR[1]                ;EXPECTED DATA (ECC DIS AND DIAG
                    ;9118                                            ; CHK SET)
U 0982, 9D45,75     ;9119          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;9120                                            ;SET UP TO READ CSR 1
U 0983, 3022,15     ;9121          MOV MEM.DATA TO WR[0]             ;GET CONTENTS OF CSR 1.  RDS BIT SHOULD
                    ;9122                                            ; HAVE BEEN CLEARED BY WRITE TO CSR 2
U 0984, 0869,3C     ;9123          JSR [CHECK.RESULT]                ;CHECK RESULT FOR ERRORS
U 0985, 0898,04     ;9124          JMP [LOOP.TD.B]                   ;ERROR LOOP IF ENABLED
                    ;9125
                    ;9126    TEST.TD.C:
U 0986, FF82,15     ;9127          INC LS[ERROR.NUMBER]              ;ERROR C
U 0987, 3673,95     ;9128          MOV LS[ECC.DIS] TO WR[3]          ;SET ECC DISABLE IN WR 3
U 0988, 4775,95     ;9129          BIS LS[DIAG.CHK] TO WR[3]         ;ALSO SET DIAG CHK IN WR 3.
U 0989, 4779,95     ;9130          BIS LS[INH.CRD] TO WR[3]          ;NOW WR 3 IS EXPECTED DATA FOR ERROR 8
U 098A, 3704,15     ;9131          MOV LS[CKBTS.1100100] TO WR[0]    ;PUT CKBTS FOR A SINGLE BIT ERROR IN WR 0
U 098B, A0C0,15     ;9132          COM WR[0]                         ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 098C, 452C,15     ;9133          BIC LS[#FFFFFF00] TO WR[0]        ;CLEAR ALL BUT LOW BYTE
U 098D, 2EC6,15     ;9134          BIS WR[3] TO WR[0]                ;SET INHIBIT REPORT CRD, DIAG CHK AND
                    ;9135                                            ; ECC DIS IN WR 0
U 098E, 8A17,FC     ;9136          JSR [WRITE.CSR1]                  ;WRITE CSR 1 WITH DATA FROM WR 0
                    ;9137
                    ;9138    LOOP.TD.C:
U 098F, 9D9C,F5     ;9139          MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                    ;9140                                            ;SET UP TO WRITE CKBTS FROM CSR 1 AND
                    ;9141                                            ; DATA FROM LS
U 0990, B29C,15     ;9142          WRITE.MEM LS[ZERO]                ;WRITE ECC DATA OF 0
U 0991, 3022,15     ;9143          MOV MEM.DATA TO WR[0]             ;COMPLETE THE CYCLE BUT DO NOT CHECK
                    ;9144
U 0992, 2006,95     ;9145          MOV WR[3] TO WR[1]                ;EXPECTED DATA (INH CRD, ECC DIS AND
                    ;9146                                            ; DIAG CHK SET)
U 0993, 9D45,75     ;9147          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;9148                                            ;SET UP TO READ CSR 1
U 0994, 3022,15     ;9149          MOV MEM.DATA TO WR[0]             ;GET CONTENTS OF CSR 1
U 0995, 0869,3C     ;9150          JSR [CHECK.RESULT]                ;CHECK RESULT FOR ERRORS
U 0996, 0898,F4     ;9151          JMP [LOOP.TD.C]                   ;ERROR LOOP IF ENABLED
                    ;9152
U 0997, FF82,15     ;9153          INC LS[ERROR.NUMBER]              ;ERROR D
                    ;9154    LOOP.TD.D:
U 0998, 5B00,1D     ;9155          SKIP.IF[MEM.REF.OK]               ;CHECK SKIP LOGIC.  THIS INSTRUCTION
                    ;9156                                            ; SHOULD NOT SKIP
U 0999, 0899,B4     ;9157          JMP [TEST.TD.E]                   ;NO ERROR IF HERE, GO TO NEXT TEST
U 099A, 889A,BC     ;9158          JSR [ERROR.TD.D]                  ;REPORT SKIP ON ERROR SUM FAILURE IF HERE
```

```
                ;9159
                ;9160   TEST.TD.E:
U 099B, FF82,15 ;9161           INC LS[ERROR.NUMBER]                ;ERROR E
                ;9162   LOOP.TD.E:
U 099C, 9D9C,F5 ;9163           MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                ;9164                                               ;SET UP TO WRITE CKBTS FROM CSR 1 AND
                ;9165                                               ; DATA FROM LS
U 099D, B29C,15 ;9166           WRITE.MEM LS[ZERO]                  ;WRITE ECC DATA OF 0
                ;9167           MOV MEM.DATA TO WR[0],              ;COMPLETE THE CYCLE, LOOK FOR ERROR SUM
U 099E, B022,1D ;9168           SKIP.IF[MEM.REF.OK]                 ;CHECK SKIP LOGIC.  THIS INSTRUCTION
                ;9169                                               ; SHOULD NOT SKIP
U 099F, 889B,D4 ;9170           JMP [END.TD]                        ;NO ERROR IF HERE, END OF TEST
U 09A0, 889A,EC ;9171           JSR [ERROR.TD.E]                    ;REPORT SKIP ON ERROR SUM FAILURE IF HERE
U 09A1, 889B,D4 ;9172           JMP [END.TD]                        ;ERROR BUT NO ERROR LOOP,  DONE
                ;9173
                ;9174
                ;9175   ERROR.TD.3:
U 09A2, 089B,1C ;9176           JSR [REPORT.TD.SKIP.ERROR]          ;FORCE ERROR REPORT ON SKIP ERROR
U 09A3, 8893,94 ;9177           JMP [LOOP.TD.3]                     ;ERROR LOOP IF RETURN HERE
U 09A4, 5B00,14 ;9178           RETURN                              ;NO ERROR LOOP RETURN
                ;9179
                ;9180   ERROR.TD.6:
U 09A5, 089B,1C ;9181           JSR [REPORT.TD.SKIP.ERROR]          ;FORCE ERROR REPORT ON SKIP ERROR
U 09A6, 0895,44 ;9182           JMP [LOOP.TD.6]                     ;ERROR LOOP IF RETURN HERE
U 09A7, 5B00,14 ;9183           RETURN                              ;NO ERROR LOOP WILL CONTINUE TESTING
                ;9184
                ;9185   ERROR.TD.A:
U 09A8, 889B,3C ;9186           JSR [REPORT.TD.SKIP.ERROR.NO.CPU] ;FORCE ERROR REPORT ON SKIP ERROR
U 09A9, 8897,B4 ;9187           JMP [LOOP.TD.A]                     ;ERROR LOOP IF RETURN HERE
U 09AA, 5B00,14 ;9188           RETURN                              ;NO ERROR LOOP WILL CONTINUE TESTING
                ;9189
                ;9190   ERROR.TD.D:
U 09AB, 889B,3C ;9191           JSR [REPORT.TD.SKIP.ERROR.NO.CPU] ;FORCE ERROR REPORT ON SKIP ERROR
U 09AC, 0899,84 ;9192           JMP [LOOP.TD.D]                     ;ERROR LOOP IF RETURN HERE
U 09AD, 5B00,14 ;9193           RETURN                              ;NO ERROR LOOP WILL CONTINUE TESTING
                ;9194
                ;9195   ERROR.TD.E:
U 09AE, 889B,3C ;9196           JSR [REPORT.TD.SKIP.ERROR.NO.CPU] ;FORCE ERROR REPORT ON SKIP ERROR
U 09AF, 8899,C4 ;9197           JMP [LOOP.TD.E]                     ;ERROR LOOP IF RETURN HERE
U 09B0, 5B00,14 ;9198           RETURN                              ;NO ERROR LOOP WILL CONTINUE TESTING
                ;9199
                ;9200
                ;9201   REPORT.TD.SKIP.ERROR:
U 09B1, 3642,15 ;9202           MOV LS[CPU] TO WR[0]                ;ADD CPU MODULE TO MODULE PRINTOUT
U 09B2, 6D8C,15 ;9203           BIS WR[0] TO LS[MODULE.NUM]         ;UPDATE MODULE CODE IN LS
                ;9204   REPORT.TD.SKIP.ERROR.NO.CPU:
U 09B3, B66E,15 ;9205           MOV LS[NA.EXP.REC] TO WR[0]         ;SET BIT 23 IN WR 0 TO INDICATE NOT TO
                ;9206                                               ; PRINT EXPECTED OR RECEIVED DATA IF AN
                ;9207                                               ; ERROR OCCURS
U 09B4, 3E80,15 ;9208           MOV WR[0] TO LS[CONTROL.STATUS]     ;UPDATE CONTROL AND STATUS WORD
U 09B5, 2F80,15 ;9209           CLR WR[0]                           ;CLEAR WR 0
U 09B6, 369E,95 ;9210           MOV LS[ONES] TO WR[1]               ;SET WR 1.  NOW ERROR ROUTINE WILL
                ;9211                                               ; REPORT AN ERROR (BUT EXP AND REC WILL
                ;9212                                               ; NOT BE PRINTED)
U 09B7, 0869,3C ;9213           JSR [CHECK.RESULT]                  ;REPORT THE ERROR
```

```
U 09B8, 5B00,14 ;9214           RETURN                          ;ERROR LOOP RETURN
U 09B9, 3644,15 ;9215           MOV LS[MCT] TO WR[0]            ;SET UP MODULE CODE. BIT 2 SET IN WR
U 09BA, 3E8C,15 ;9216           MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                ;9217                                           ; MCT BOARD ONLY
U 09BB, E580,15 ;9218           CLR LS[CONTROL.STATUS]          ;REENABLE EXP AND REC PRINTOUT
U 09BC, DB00,16 ;9219           RETURN+1                        ;NO ERROR LOOP RETURN
                ;9220
                ;9221   END.TD:
```

```
;9222    .PAGE   "  TEST E ECC Correction Test (M8391  MCT Module)"
;9223    ;++
;9224    ;
;9225    ; Test Description:
;9226    ;
;9227    ;       This test checks the ECC correction logic to assure that any bit can
;9228    ;       be corrected from a 1 to a 0 and from a 0 to a 1.  It also verifies
;9229    ;       the MCT branch on error and branch on single error logic (BEN0=ERROR.L
;9230    ;       and BEN2=SINGLE.ERR.L).  The CSR will be loaded with CKBTS which cor-
;9231    ;       respond with ECC data of all 0's or all 1's.  Then data of shifting 1's
;9232    ;       and shifting 0's is input into the ECC data latches and the resulting
;9233    ;       data is checked for correction to all 0's or all 1's respectively.
;9234    ;       Double bit errors will be tested also to assure that detection but no
;9235    ;       correction of data takes place.  The data path is as follows:  A
;9236    ;       MEM.REQ is issued with MF field=MAINT.ECC.DATA.  The LS address used
;9237    ;       contains data of 0(H) for branching later in the test.  This data is
;9238    ;       loaded into the VAR and a dispatch is made to the proper routine (see
;9239    ;       dispatch description at the beginning of this listing).  At the dis-
;9240    ;       patch address, memory busy and rotate disable are set, the PA bits are
;9241    ;       enabled and transceivers on the CPU module are enabled to allow data
;9242    ;       from the CPU to be received.  In the next cycle, the data rotators are
;9243    ;       clocked to disable rotation and the events of the previous cycle are
;9244    ;       repeated (except SET ADDR PH is not needed).  The following cycle opens
;9245    ;       the ECC latches, enables the output of the data rotators on the ARRAY
;9246    ;       BUS, clears any errors in the CSR, keeps memory busy set, and branches
;9247    ;       on L CPU DR.  CPU DR (data request) comes up when the CPU does a MOV u-
;9248    ;       instruction.  In this case, the CPU is sending the test data pattern
;9249    ;       for the rotator PALs.  If L CPU DR is not yet asserted, memory busy is
;9250    ;       dropped and the memory u-code will loop waiting for it.  Once L CPU DR
;9251    ;       is asserted, the next u-cycle will set 2nd memory cycle, open the ECC
;9252    ;       latches, enable the rotator PAL outputs onto the ARRAY BUS, and keep
;9253    ;       the CPU transceivers enabled.  Setting 2nd memory cycle keeps the data,
;9254    ;       sent over the MC BUS from the CPU, latched inside the rotator PAL (the
;9255    ;       data was loaded in the last u-cycle).  The next cycle keeps the ECC
;9256    ;       latches open, keeps the rotator PAL output on the ARRAY BUS, and waits
;9257    ;       for D DATA REC to be asserted. DATA RCVD is asserted by the CPU at the
;9258    ;       end (P2) of the CPU MOV instruction.  Looping here until the signal is
;9259    ;       asserted before asserting memory busy assures that the CPU will move on
;9260    ;       to the next u-instruction.  When D DATA REC is asserted, the memory u-
;9261    ;       code will branch on LVA00 (which was cleared at the beginning of the
;9262    ;       CPU MEM.REQ instruction) to the proper test.  This u-cycle will assert
;9263    ;       memory busy, open the ECC latches, enable the rotator PAL outputs onto
;9264    ;       the ARRAY BUS, and enable the checkbits from CSR 1 onto the checkbit
;9265    ;       ARRAY BUS.  It will then branch to the proper test according to the
;9266    ;       state of ECC DIS and DIAG CHK in CSR 1.  CSR 1 was loaded prior to
;9267    ;       issueing the MEM.REQ instruction.  For this test, DIAG.CHK is set and
;9268    ;       ECC.DIS is clear.  After the branch, 2ND MEMORY CYCLE is cleared and
;9269    ;       memory busy is left asserted.  A branch is performed on ERROR.L also.
;9270    ;       If there is no error, the ECC data is placed on the MC BUS and the
;9271    ;       memory loops waiting for a CPU DATA REQ.  Then it waits for CPU DATA
;9272    ;       RCVD after which it goes to the idle loop.  If there is an error, the
;9273    ;       next cycle clocks RDS or CRD in CSR 1, enables the syndrome bits from
;9274    ;       high word ECC chip onto the CB BUS, and branches on single error.  If
;9275    ;       a double error was detected the syndromes are latched into CSR 0 and
;9276    ;       the data from the ECC chips is read by the CPU as above.  If a single
```

```
;9277    ;       error occurred, the next cycle enables correction, outputs the syndrome
;9278    ;       bits onto the CB BUS and opens the ECC output data latch.  The follow-
;9279    ;       cycle repeats that and also clocks CSR 0 with the syndrome bits.  The
;9280    ;       next cycle closes the ECC output data latch, keeping correction enabled
;9281    ;       and syndromes enabled to avoid any glitches.  The result is then read
;9282    ;       by the CPU as before and checked.  The data patterns used are all 1's
;9283    ;       and all zeros with CKBTS set up to produce no error.  This is followed
;9284    ;       by shifting 1's and shifting zeros with CKBTs set to correct the shift-
;9285    ;       ing bit.
;9286    ;
;9287    ; Assumptions:
;9288    ;
;9289    ;       It is assumed that all previous tests have run successfully.
;9290    ;
;9291    ; Test Steps:
;9292    ;
;9293    ;       1)  Set up error mask, error number, and module number in LS (for
;9294    ;           error printouts) and clear previous error number in LS.
;9295    ;       2)  Write CSR 0 to 1's by executing CKBT test.  Then write CSR 1 to set
;9296    ;           DIAG CHK, clear ECC DIS, and write complement of CKBTs for data of
;9297    ;           all 1's or all 0's (complement of 0111100(B) from CB T to CB 1
;9298    ;           respectively).
;9299    ;       3)  Load WR 1 with data pattern from LS (expected ECC data).
;9300    ;       4)  Issue a MEM.REQ with DT=longword (11) and MF=MAINT.ECC.DATA using
;9301    ;           a location in LS that contains a 0(H) as the address (this is load-
;9302    ;           ed into the VAR and used to branch later).
;9303    ;       5)  Execute a WRITE.MEM LS with current ECC data pattern.
;9304    ;       6)  Execute a MOVE MEM.DATA TO WR[0] and check that data was not alter-
;9305    ;           ed.
;9306    ;       7)  Change error mask to check bits 6-0, load WR 1 with expected CSR 0
;9307    ;           result (ones), read CSR 0, and check.  This assures that the right
;9308    ;           branch was taken on branch on ERROR.L.
;9309    ;       8)  Change error mask to check bits 31 and 30 of CSR 1 (RDS and CRD).
;9310    ;           Read CSR 1 and check that RDS and CRD bits are clear.  Execute skip
;9311    ;           on ERROR SUM NOT in CPU, checking for a skip.
;9312    ;       9)  Restore error mask to 32 bits and repeat steps 2 through 6 with
;9313    ;           next data pattern (0's).
;9314    ;       10) Repeat steps 2 through 5 with shifting 1's data pattern.
;9315    ;       11) Execute a MOV MEM.DATA TO WR[0] and check for all 0's (corrected
;9316    ;           data).  Repeat steps 8 and 10 until a one has been shifted across
;9317    ;           all bits. Check for CRD set and a skip on ERROR SUM in step 8.
;9318    ;       12) Repeat step 10 with shifting 0's data.  Execute a MOV MEM.DATA TO
;9319    ;           WR[0] and check for all ones (corrected data).
;9320    ;       13) Repeat steps 2 through 5 with double error data pattern as ECC
;9321    ;           data.  Double error patterns are 3(H) and 30000(H) (double error in
;9322    ;           low word and high word respectively).
;9323    ;       14) Execute a MOV MEM.DATA TO WR[0] and check for no change (double
;9324    ;           errors are not corrected).
;9325    ;       15) Repeat step 8 checking for RDS set, CRD clear and no skip on ERROR
;9326    ;           SUM NOT.
;9327    ;       16) Set INH REP CRD in CSR 1 and check for a skip and no CRD set with
;9328    ;           a single error.
;9329    ;       17) Set INH REP CDR in CSR 1 and check for no skip and RDS set with a
;9330    ;           double bit error.
;9331    ;
```

```
;9332   ; Errors:
;9333   ;
;9334   ;       NOTE:  Expected and Received data is as follows:  ECC data when error
;9335   ;              mask is all 0's, CSR 0 data when error mask is FFFFFF80, and
;9336   ;              CSR 1 data when error mask is 3FFFFFFF.
;9337   ;
;9338   ;       Error 1 - ECC data altered with CKBTs set for no error (data of 1's).
;9339   ;       Error 2 - CKBTs in ECC chips altered on no error condition or error
;9340   ;                 branch failed.
;9341   ;       Error 3 - RDS or CRD bit set in CSR 1 on no error condition.
;9342   ;       Error 4 - CPU skip on ERROR SUM NOT failed to skip with no error (MCT
;9343   ;                 module at fault).
;9344   ;       Error 5 - ECC data altered with CKBTs set for no error (data of 0's).
;9345   ;       Error 6 - ECC logic failed to correct single error from a 1 to a 0.
;9346   ;                 OTHER data is data pattern being corrected to all 0's.
;9347   ;       Error 7 - RDS set and/or CRD clear after single bit error.
;9348   ;                 OTHER data is data pattern being corrected to all 0's.
;9349   ;       Error 8 - CPU skip on ERROR SUM NOT skipped after single bit error.
;9350   ;       Error 9 - ECC logic failed to correct single error from a 0 to a 1.
;9351   ;                 OTHER data is data pattern being corrected to all 1's.
;9352   ;       Error A - ECC logic altered data on double bit error or branch on sin-
;9353   ;                 gle bit error failed with data of 3(H).
;9354   ;       Error B - RDS clear and/or CRD set on double bit error with data of
;9355   ;                 3(H).
;9356   ;       Error C - CPU skip on ERROR SUM NOT skipped after double bit error.
;9357   ;       Error D - ECC logic altered data on double bit error or branch on sin-
;9358   ;                 gle bit error failed with data of 30000(H).
;9359   ;       Error E - RDS clear and/or CRD set on double bit error with data of
;9360   ;                 30000(H).
;9361   ;       Error F - CRD or RDS in CSR 1 set on single bit error with INH REP CRD set.
;9362   ;       Error 10- CPU skip on ERROR SUM NOT failed to skip on single bit error
;9363   ;                 with INH REP CRD set.
;9364   ;       Error 11- RDS clear or CRD set after double error with INH REP CRD set.
;9365   ;       Error 12- CPU skip on ERROR SUM NOT skipped on double error with INH
;9366   ;                 REP CRD set.
;9367   ;
;9368   ; Debug:
;9369   ;
;9370   ;       Two methods of debug will be explained.  If the MCT module is in a
;9371   ;       test station then the MCT can be single stepped making debug easier
;9372   ;       in some cases.  Otherwise only the CPU can be single stepped and this
;9373   ;       method will be explained also.  When single stepping the memory con-
;9374   ;       troller will help in debug, this section will explicitly suggest it.
;9375   ;
;9376   ;       Note:  The data written into the ECC chips data portion is printed un-
;9377   ;       der 'OTHER' in the error report on shifted 1's and shifted 0 patterns.
;9378   ;
;9379   ;       Error 1 - This error indicates a problem with the ECC chip itself.  It
;9380   ;       is not readily apparent which ECC chip is failing, but all the input
;9381   ;       lines except CORR DIS L being high have been checked.  If CORR DIS L is
;9382   ;       high at both ECC chips, suspect one of those chips.
;9383   ;
;9384   ;       Error 2 - This error is most likely to occur as a result of the branch
;9385   ;       on ERROR L failing in the no error direction.  If branch on error takes
;9386   ;       the error path, the syndromes from the ECC chip will be written into
```

```
;9387    ;       CSR 0.  These will be different from the CKBITs originally written into
;9388    ;       CSR 0.  If MCT single step is available, single step the CPU to the
;9389    ;       WRITE.MEM LS instruction, enable MCT single step, step the CPU to the
;9390    ;       MOV MEM.DATA TO WR[0], and step the MCT to the branch on ERROR L (BEN 0=
;9391    ;       ERROR.L).  Check ERROR L at the BEN 0 mux for a high and the select for
;9392    ;       a 011(B) on SEL 4 to SEL 1 respectively.  If OK, step the MCT again and
;9393    ;       check for no branch.  If a branch occurs, the BEN 0 mux is bad.
;9394    ;
;9395    ;       Error 3 - This error most likely indicates a problem with the error
;9396    ;       branch or the ECC chip.  If the ECC chip is OK, the no error condition
;9397    ;       should have bypassed the CSR ERR ADR CLK necessary to set bits in the
;9398    ;       CSR.  If the ECC chip failed, anything could have happened to the CRD
;9399    ;       or RDS bits.
;9400    ;
;9401    ;       Error 4 - If this is the first error, suspect the CSR CONTROL PAL.
;9402    ;
;9403    ;       Error 5 - Same as error 1.
;9404    ;
;9405    ;       Error 6 - This error indicates a problem with one of the ECC chips or
;9406    ;       the branch logic on SINGLE ERR L.  Check the signals SINGLE ERR L and
;9407    ;       ERROR L out of the high word ECC chip for lows.  This can be checked
;9408    ;       after the console has halted on error.  If either of these is bad, then
;9409    ;       one of the ECC chips is faulty.  If these are both low, another possi-
;9410    ;       bility is that the branch on SINGLE ERR is bad.  If the branch fails,
;9411    ;       CORR DIS L will not go high, and no correction will occur.  First check
;9412    ;       the signal SINGLE ERR L at the BEN 2 mux for a low.  If OK, and MCT
;9413    ;       single step is available, step the CPU to the WRITE.MEM LS instruction,
;9414    ;       enable MCT single step, step the CPU to the MOV MEM.DATA TO WR[0], and
;9415    ;       step MCT to the branch on SINGLE ERR (BEN 2=SINGLE.ERR.L).  Check the
;9416    ;       signal SINGLE ERR L at the BEN 2 MUX and the select lines for a 011(B)
;9417    ;       on SEL 4 through SEL 1 respectively.  If these are OK, step the MCT
;9418    ;       once more.  It should end up at the cycle that brings CORR DIS L high.
;9419    ;       If the branch fails, the BEN 2 MUX is bad.  Otherwise check that CORR
;9420    ;       DIS L is high at both ECC chips.  If OK, one of the ECC chips is prob-
;9421    ;       ably bad.  If MCT single step is not available, check CORR DIS L while
;9422    ;       looping on error.
;9423    ;
;9424    ;       Error 7 - If this is the first error, one possibility is that SINGLE
;9425    ;       ERR L is not being asserted by the high word ECC chip, but that the
;9426    ;       branch on SINGLE ERR L (BEN 2=SINGLE.ERR.L) also failed.  The other
;9427    ;       logic used to control the CSR CONTROL PAL has been previously tested.
;9428    ;       If MCT single step is available, check the inputs to the CSR CONTROL
;9429    ;       PAL during the cycle that asserts CSR ERR ADDR CLK A L.  The inputs
;9430    ;       should be a low on CSR ERR ADDR CLK A L, SINGLE ERR L, ERROR L, CSR
;9431    ;       ERR SUM CLK H and INH REP CRD H.  Check WR CSR L and CPH/UBL for a
;9432    ;       high.  A bad bit in the control store could be causing the problem
;9433    ;       also.
;9434    ;
;9435    ;       Error 8 - The most likely suspect if this error occurs is the CSR CON-
;9436    ;       TROL PAL itself.
;9437    ;
;9438    ;       Error 9 - The most likely suspect, if this is the first error, is one
;9439    ;       of the ECC chips.
;9440    ;
;9441    ;       Error A - This error indicates a problem with one of the ECC chips or
```

```
                    ;9442   ;       the branch on SINGLE ERR logic.  Check the signal ERROR L at the high
                    ;9443   ;       word ECC chip for a low, and the signal SINGLE ERR L for a high.  This
                    ;9444   ;       can be done after the console halts on error.  If either of these is
                    ;9445   ;       incorrect, then one of the ECC chips is bad.  If these are OK, check
                    ;9446   ;       the signal SINGLE ERR L at the BEN 2 MUX for a high.  If this is OK,
                    ;9447   ;       and MCT single step is available, follow the procedure in error 6 above
                    ;9448   ;       lcoking for no branch.  If the u-code ends up setting DIS CORR L high,
                    ;9449   ;       then the BEN 2 MUX is probably bad.  Otherwise one of the ECC chips is
                    ;9450   ;       the most likely suspect.
                    ;9451   ;
                    ;9452   ;       Error B - The most likely suspect is one of the ECC chips.  The branch
                    ;9453   ;       on ERROR L may have taken the no error path because the ECC chip did
                    ;9454   ;       not assert ERROR L.  This would have left the data unchanged, but fail-
                    ;9455   ;       ed to clock CSR 1 to set RDS.
                    ;9456   ;
                    ;9457   ;       Error C - The most likely suspect if this error occurs is the CSR CON-
                    ;9458   ;       TROL PAL itself.
                    ;9459   ;
                    ;9460   ;       Error D - Suspect one of the ECC chips.
                    ;9461   ;
                    ;9462   ;       Error E - Same as error 11.
                    ;9463   ;
                    ;9464   ;       Error F to 12 - These errors indicate a problem with the CSR CONTROL
                    ;9465   ;       PAL.
                    ;9466   ;
                    ;9467   ;
                    ;9468   ;--
                    ;9469
                    ;9470   T.E:
U 09BD, B65E,15     ;9471           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;9472                                           ; STATUS WORD
U 09BE, 3E80,15     ;9473           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;9474                                           ; BIT 15 INDICATES START OF TEST TO
                    ;9475                                           ; CONSOLE
U 09BF, 10E0,15     ;9476           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;9477   WAIT.TE.0:
U 09C0, 889C,04     ;9478           JMP [WAIT.TE.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                    ;9479                                           ; RESPOND
                    ;9480
U 09C1, 0A1A,AC     ;9481           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;9482
                    ;9483   WRITE.CSR0:
U 09C2, 3672,15     ;9484           MOV LS[ECC.DIS] TO WR[0]        ;SET ECC DISABLE BIT IN WR
U 09C3, 4774,15     ;9485           BIS LS[DIAG.CHK] TO WR[0]       ;ALSO SET DIAG CHK BIT
U 09C4, C726,15     ;9486           BIS LS[#FF] TO WR[0]            ;ALSO SET CKBTS
U 09C5, 8A17,FC     ;9487           JSR [WRITE.CSR1]                ;SET ECC DIS AND DIAG CHK BITS IN CSR 1
                    ;9488                                           ; SET CKBT PATTERN IN BITS 6-0
U 09C6, 9D9C,F5     ;9489           MEM.REQ[MAINT.ECC.DATA] ADRS[ZERO] DT[LONG]
                    ;9490                                           ;EXECUTE TEST TO WRITE ONES IN CKBTS
                    ;9491                                           ; (NOW CSR 0 CONTAINS 1'S)
U 09C7, 329E,15     ;9492           WRITE.MEM LS[ONES]              ;WRITE DATA OF ONES IN ECC CHIP.  CKBTS
                    ;9493                                           ; GET DATA OF 1'S FROM CSR 1
U 09C8, 3022,15     ;9494           MOV MEM.DATA TO WR[0]           ;READ RESULT TO COMPLETE U-CYCLE
                    ;9495
U 09C9, B700,15     ;9496           MOV LS[CKBTS.0111100] TO WR[0]  ;SET UP CKBTS FOR DATA OF ALL 0'S OR
```

```
                    ;9497                                   ; ALL 1'S
U 09CA, A0C0,15     ;9498           COM WR[0]               ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 09CB, 452C,15     ;9499           BIC LS[#FFFFFF00] TO WR[0]  ;CLEAR ALL BUT LOW BYTE
U 09CC, 4774,15     ;9500           BIS LS[DIAG.CHK] TO WR[0]   ;SET DIAG CHK BIT IN WR 0
U 09CD, 8A17,FC     ;9501           JSR [WRITE.CSR1]        ;WRITE CSR 1 WITH DATA FROM WR 0
                    ;9502
U 09CE, 3713,15     ;9503           MOV LS[#FFFFFF80] TO WR[2]  ;SET UP WR 2 WITH ERROR MASK TO CHECK
                    ;9504                                   ; BITS 6-0
                    ;9505   LOOP.TE.1:
U 09CF, B640,15     ;9506           MOV LS[#1] TO WR[0]     ;1 IN WR
U 09D0, BE82,15     ;9507           MOV WR[0] TO LS[ERROR.NUMBER]  ;SET UP ERROR NUMBER IN LS
U 09D1, E58A,15     ;9508           CLR LS[ERROR.MASK]      ;CHECK ALL BITS
U 09D2, 369E,95     ;9509           MOV LS[ONES] TO WR[1]   ;EXPECTED DATA
U 09D3, 9D9C,F5     ;9510           MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                    ;9511                                   ;SET UP TO WRITE CKBTS FROM CSR 1, DATA
                    ;9512                                   ; FROM LS
U 09D4, 329E,15     ;9513           WRITE.MEM LS[ONES]      ;WRITE DATA OF ONES IN ECC CHIP
U 09D5, 3022,15     ;9514           MOV MEM.DATA TO WR[0]   ;READ DATA BACK (SHOULD BE NO CHANGE)
U 09D6, 0869,3C     ;9515           JSR [CHECK.RESULT]      ;CHECK FOR ERRORS
U 09D7, 889C,F4     ;9516           JMP [LOOP.TE.1]         ;ERROR LOOP IF ENABLED
                    ;9517
                    ;9518   TEST.TE.2:
U 09D8, FF82,15     ;9519           INC LS[ERROR.NUMBER]    ;ERROR 2
U 09D9, BE88,15     ;9520           MOV WR[2] TO LS[ERROR.MASK]  ;ERROR MASK TO CHECK BITS 6-0
U 09DA, 369E,95     ;9521           MOV LS[ONES] TO WR[1]   ;EXPECTED DATA (UNMODIFIED CSR 0)
U 09DB, 9D9D,75     ;9522           MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                    ;9523                                   ;SET UP TO READ CSR 0
U 09DC, 3022,15     ;9524           MOV MEM.DATA TO WR[0]   ;GET CKBTS FROM CSR 0.  THESE SHOULD BE
                    ;9525                                   ; UNCHANGED FROM WHEN THEY WERE WRITTEN
                    ;9526                                   ; AT THE BEGINNING OF THIS TEST
U 09DD, 0869,3C     ;9527           JSR [CHECK.RESULT]      ;CHECK FOR ERRORS
U 09DE, 889C,F4     ;9528           JMP [LOOP.TE.1]         ;ERROR LOOP IF ENABLED
                    ;9529
                    ;9530   TEST.TE.3:
U 09DF, FF82,15     ;9531           INC LS[ERROR.NUMBER]    ;ERROR 3
U 09E0, 5F7E,15     ;9532           MCOM LS[RDS] TO WR[0]   ;CLEAR BIT 31 IN WR 0, SET OTHERS
U 09E1, 457C,15     ;9533           BIC LS[CRD] TO WR[0]    ;CLEAR BIT 30 IN WR 0
U 09E2, 3E8A,15     ;9534           MOV WR[0] TO LS[ERROR.MASK]  ;SET UP TO CHECK BITS 30 AND 31
U 09E3, 2F82,95     ;9535           CLR WR[1]               ;EXPECTED DATA (BOTH BITS CLEAR)
U 09E4, 9D45,75     ;9536           MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;9537                                   ;SET UP TO READ CSR 1
U 09E5, 3022,15     ;9538           MOV MEM.DATA TO WR[0]   ;GET RESULT
U 09E6, 0869,3C     ;9539           JSR [CHECK.RESULT]      ;CHECK FOR ERRORS
U 09E7, 889C,F4     ;9540           JMP [LOOP.TE.1]         ;ERROR LOOP IF ENABLED
                    ;9541
                    ;9542   TEST.TE.4:
U 09E8, FF82,15     ;9543           INC LS[ERROR.NUMBER]    ;ERROR 4
                    ;9544
U 09E9, 5B00,1D     ;9545           SKIP.IF[MEM.REF.OK]     ;SKIP IF NO MEMORY ERROR (SHOULD SKIP)
U 09EA, 08A7,4C     ;9546           JSR [ERROR.TE.4]        ;ERROR IF HERE. GO REPORT IT
                    ;9547
                    ;9548   TEST.TE.5:
U 09EB, FF82,15     ;9549           INC LS[ERROR.NUMBER]    ;ERROR 5
U 09EC, E58A,15     ;9550           CLR LS[ERROR.MASK]      ;CHECK ALL BITS
                    ;9551   LOOP.TE.5:
```

```
U 09ED, B69C,95 ;9552          MOV LS[ZERO] TO WR[1]              ;EXPECTED DATA
U 09EE, 9D9C,F5 ;9553          MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                ;9554                                            ;SET UP TO WRITE CKBTS FROM CSR 1, DATA
                ;9555                                            ; FROM LS
U 09EF, B29C,15 ;9556          WRITE.MEM LS[ZERO]                 ;WRITE DATA OF ZEROS IN ECC CHIP
U 09F0, 3022,15 ;9557          MOV MEM.DATA TO WR[0]              ;READ DATA BACK (SHOULD BE NO CHANGE)
U 09F1, 0869,3C ;9558          JSR [CHECK.RESULT]                 ;CHECK FOR ERRORS
U 09F2, 889E,D4 ;9559          JMP [LOOP.TE.5]                    ;ERROR LOOP IF ENABLED
                ;9560
                ;9561  TEST.TE.6:
U 09F3, FF82,15 ;9562          INC LS[ERROR.NUMBER]               ;ERROR 6
U 09F4, 3683,95 ;9563          MOV LS[ERROR.NUMBER] TO WR[3]      ;STORE ERROR NUMBER
U 09F5, DF7F,15 ;9564          MCOM LS[RDS] TO WR[2]              ;CLEAR BIT 31 IN WR 2, SET OTHERS
U 09F6, C57D,15 ;9565          BIC LS[CRD] TO WR[2]               ;CLEAR BIT 30 IN WR 2. USED AS ERROR
                ;9566                                            ; MASK FOR ERROR 7
U 09F7, E5F8,15 ;9567          CLR LS[OS]                         ;CLEAR INDEX
                ;9568  REPEAT.TE.6:
U 09F8, 36F6,15 ;9569          MOV LS[SHIFT.OS(4-0)] TO WR[0]     ;GET ECC DATA PATTERN
U 09F9, BE88,15 ;9570          MOV WR[0] TO LS[ADDRESS.DATA]      ;PUT IN OTHER DATA LOCATION FOR PRINOUT
U 09FA, B670,15 ;9571          MOV LS[OTHER.DATA] TO WR[0]        ;SET BIT 24
U 09FB, 3E80,15 ;9572          MOV WR[0] TO LS[CONTROL.STATUS]    ;TELL CONSOLE TO PRINT ECC DATA PATTERN
                ;9573                                            ; UNDER 'OTHER' IN ERROR REPORT
                ;9574  LOOP.TE.6:
U 09FC, BE83,95 ;9575          MOV WR[3] TO LS[ERROR.NUMBER]      ;SET UP ERROR NUMBER IN LS
U 09FD, E58A,15 ;9576          CLR LS[ERROR.MASK]                 ;CHECK ALL BITS
U 09FE, B69C,95 ;9577          MOV LS[ZERO] TO WR[1]              ;EXPECTED DATA
U 09FF, 9D9C,F5 ;9578          MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                ;9579                                            ;SET UP TO WRITE CKBTS FROM CSR 1, DATA
                ;9580                                            ; FROM LS
U 0A00, B2F6,15 ;9581          WRITE.MEM LS[SHIFT.OS(4-0)]        ;WRITE DATA OF SHIFTING ONES IN ECC CHIP
U 0A01, 3022,15 ;9582          MOV MEM.DATA TO WR[0]              ;READ DATA BACK (SHOULD BE CORRECTED TO
                ;9583                                            ; ALL ZEROS)
U 0A02, 0869,3C ;9584          JSR [CHECK.RESULT]                 ;CHECK FOR ERRORS
U 0A03, 889F,C4 ;9585          JMP [LOOP.TE.6]                    ;ERROR LOOP IF ENABLED
                ;9586
                ;9587  TEST.TE.7:
U 0A04, FF82,15 ;9588          INC LS[ERROR.NUMBER]               ;ERROR 7
U 0A05, BE8B,15 ;9589          MOV WR[2] TO LS[ERROR.MASK]        ;SET UP TO CHECK BITS 30 AND 31
U 0A06, 367C,95 ;9590          MOV LS[CRD] TO WR[1]               ;EXPECTED DATA (CRD SET, RDS CLEAR)
U 0A07, 9D45,75 ;9591          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;9592                                            ;SET UP TO READ CSR 1
U 0A08, 3022,15 ;9593          MOV MEM.DATA TO WR[0]              ;GET RESULT
U 0A09, 0869,3C ;9594          JSR [CHECK.RESULT]                 ;CHECK FOR ERRORS
U 0A0A, 889F,C4 ;9595          JMP [LOOP.TE.6]                    ;ERROR LOOP IF ENABLED
                ;9596
U 0A0B, 7FF8,15 ;9597          INC LS[OS]                         ;INCREMENT INDEX FOR NEXT PATTERN
U 0A0C, B6F8,15 ;9598          MOV LS[OS] TO WR[0]                ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                ;9599          BIT LS[BIT5] WITH WR[0],           ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0A0D, 594A,35 ;9600           DT(LONG)&SET.ALU.CC               ; AND SET CONDITION CODES
U 0A0E, 889F,89 ;9601          JMP.IF[BITS.CLR] TO [REPEAT.TE.6] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                ;9602
                ;9603  TEST.TE.8:
U 0A0F, FF82,15 ;9604          INC LS[ERROR.NUMBER]               ;ERROR 8
                ;9605  LOOP.TE.8:
U 0A10, 5B00,1D ;9606          SKIP.IF[MEM.REF.OK]                ;SKIP IF NO MEMORY ERROR (SHOULD NOT
```

```
                ;9607                                   ; SKIP)
U 0A11, 08A1,34 ;9608           JMP [TEST.TE.9]
U 0A12, 08A7,7C ;9609           JSR [ERROR.TE.8]        ;ERROR IF HERE. GO REPORT IT
                ;9610
                ;9611   TEST.TE.9:
U 0A13, FF82,15 ;9612           INC LS[ERROR.NUMBER]    ;ERROR 9
U 0A14, E58A,15 ;9613           CLR LS[ERROR.MASK]      ;CHECK ALL BITS
U 0A15, E5F8,15 ;9614           CLR LS[OS]              ;CLEAR INDEX
                ;9615   REPEAT.TE.9:
U 0A16, 5FF6,15 ;9616           MCOM LS[SHIFT.OS(4-0)] TO WR[0] ;GET DATA PATTERN, COMPLEMENT, AND PUT
                ;9617                                   ; IN WR[0]
U 0A17, 3E10,15 ;9618           MOV WR[0] TO LS[T8]      ;STORE IN TEMPORARY LS LOCATION
U 0A18, BE88,15 ;9619           MOV WR[0] TO LS[ADDRESS.DATA] ;PUT IN OTHER DATA LOCATION FOR PRINOUT
U 0A19, B670,15 ;9620           MOV LS[OTHER.DATA] TO WR[0] ;SET BIT 24
U 0A1A, 3E80,15 ;9621           MOV WR[0] TO LS[CONTROL.STATUS] ;TELL CONSOLE TO PRINT ECC DATA PATTERN
                ;9622                                   ; UNDER 'OTHER' IN ERROR REPORT
                ;9623   LOOP.TE.9:
U 0A1B, 369E,95 ;9624           MOV LS[ONES] TO WR[1]   ;EXPECTED DATA
U 0A1C, 9D9C,F5 ;9625           MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                ;9626                                   ;SET UP TO WRITE CKBTS FROM CSR 1, DATA
                ;9627                                   ; FROM LS
U 0A1D, 3210,15 ;9628           WRITE.MEM LS[T8]        ;WRITE DATA OF SHIFTING ZEROS IN ECC CHIP
U 0A1E, 3022,15 ;9629           MOV MEM.DATA TO WR[0]   ;READ DATA BACK (SHOULD BE CORRECTED TO
                ;9630                                   ; ALL ONES)
U 0A1F, 0869,3C ;9631           JSR [CHECK.RESULT]      ;CHECK FOR ERRORS
U 0A20, 88A1,B4 ;9632           JMP [LOOP.TE.9]         ;ERROR LOOP IF ENABLED
                ;9633
U 0A21, 7FF8,15 ;9634           INC LS[OS]              ;INCREMENT INDEX FOR NEXT PATTERN
U 0A22, B6F8,15 ;9635           MOV LS[OS] TO WR[0]     ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                ;9636           BIT LS[BIT5] WITH WR[0], ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0A23, 594A,35 ;9637            DT(LONG)&SET.ALU.CC    ; AND SET CONDITION CODES
U 0A24, 88A1,69 ;9638           JMP.IF[BITS.CLR] TO [REPEAT.TE.9] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                ;9639
                ;9640   TEST.TE.A:
U 0A25, E580,15 ;9641           CLR LS[CONTROL.STATUS]  ;CLEAR CONTROL AND STATUS WORD
U 0A26, FF82,15 ;9642           INC LS[ERROR.NUMBER]    ;ERROR A
U 0A27, 3683,95 ;9643           MOV LS[ERROR.NUMBER] TO WR[3] ;STORE ERROR NUMBER
U 0A28, DF7F,15 ;9644           MCOM LS[RDS] TO WR[2]   ;CLEAR BIT 31 IN WR 2, SET OTHERS
U 0A29, C57D,15 ;9645           BIC LS[CRD] TO WR[2]    ;CLEAR BIT 30 IN WR 2. USED AS ERROR
                ;9646                                   ; MASK FOR ERROR B
                ;9647   LOOP.TE.A:
U 0A2A, BE83,95 ;9648           MOV WR[3] TO LS[ERROR.NUMBER] ;SET UP ERROR NUMBER IN LS
U 0A2B, E58A,15 ;9649           CLR LS[ERROR.MASK]      ;CHECK ALL BITS
U 0A2C, B6C0,95 ;9650           MOV LS[#3(H)] TO WR[1]  ;EXPECTED DATA
U 0A2D, 9D9C,F5 ;9651           MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                ;9652                                   ;SET UP TO WRITE CKBTS FROM CSR 1, DATA
                ;9653                                   ; FROM LS
U 0A2E, B2C0,15 ;9654           WRITE.MEM LS[#3(H)]     ;WRITE DATA OF 3(H) IN ECC CHIP (DOUBLE
                ;9655                                   ; ERROR)
U 0A2F, 3022,15 ;9656           MOV MEM.DATA TO WR[0]   ;READ DATA BACK (SHOULD NOT BE MODIFIED)
U 0A30, 0869,3C ;9657           JSR [CHECK.RESULT]      ;CHECK FOR ERRORS
U 0A31, 08A2,A4 ;9658           JMP [LOOP.TE.A]         ;ERROR LOOP IF ENABLED
                ;9659
                ;9660   TEST.TE.B:
U 0A32, FF82,15 ;9661           INC LS[ERROR.NUMBER]    ;ERROR B
```

```
U 0A33, BE8B,15 ;9662          MOV WR[2] TO LS[ERROR.MASK]     ;SET UP TO CHECK BITS 30 AND 31
U 0A34, B67E,95 ;9663          MOV LS[RDS] TO WR[1]            ;EXPECTED DATA (RDS SET, CRD CLEAR)
U 0A35, 9D45,75 ;9664          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;9665                                         ;SET UP TO READ CSR 1
U 0A36, 3022,15 ;9666          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0A37, 0869,3C ;9667          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0A38, 08A2,A4 ;9668          JMP [LOOP.TE.A]                 ;ERROR LOOP IF ENABLED
                ;9669
                ;9670  TEST.TE.C:
U 0A39, FF82,15 ;9671          INC LS[ERROR.NUMBER]            ;ERROR C
U 0A3A, 5B00,1D ;9672          SKIP.IF[MEM.REF.OK]             ;SKIP IF NO MEMORY ERROR (SHOULD NOT
                ;9673                                         ; SKIP)
U 0A3B, 08A3,D4 ;9674          JMP [TEST.TE.D]
U 0A3C, 88A7,AC ;9675          JSR [ERROR.TE.C]                ;ERROR IF HERE. GO REPORT IT
                ;9676
                ;9677  TEST.TE.D:
U 0A3D, FF82,15 ;9678          INC LS[ERROR.NUMBER]            ;ERROR D
U 0A3E, 3683,95 ;9679          MOV LS[ERROR.NUMBER] TO WR[3]   ;STORE ERROR NUMBER
U 0A3F, DF7F,15 ;9680          MCOM LS[RDS] TO WR[2]           ;CLEAR BIT 31 IN WR 2, SET OTHERS
U 0A40, C57D,15 ;9681          BIC LS[CRD] TO WR[2]            ;CLEAR BIT 30 IN WR 2. USED AS ERROR
                ;9682                                         ; MASK FOR ERROR E
                ;9683  LOOP.TE.D:
U 0A41, BE83,95 ;9684          MOV WR[3] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
U 0A42, E58A,15 ;9685          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
U 0A43, B734,95 ;9686          MOV LS[#30000] TO WR[1]         ;EXPECTED DATA
U 0A44, 9D9C,F5 ;9687          MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                ;9688                                         ;SET UP TO WRITE CKBTS FROM CSR 1, DATA
                ;9689                                         ; FROM LS
U 0A45, B334,15 ;9690          WRITE.MEM LS[#30000]            ;WRITE DATA OF 30000(H) IN ECC CHIP
                ;9691                                         ; (DOUBLE ERROR)
U 0A46, 3022,15 ;9692          MOV MEM.DATA TO WR[0]           ;READ DATA BACK (SHOULD NOT BE MODIFIED)
U 0A47, 0869,3C ;9693          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0A48, 88A4,14 ;9694          JMP [LOOP.TE.D]                 ;ERROR LOOP IF ENABLED
                ;9695
                ;9696  TEST.TE.E:
U 0A49, FF82,15 ;9697          INC LS[ERROR.NUMBER]            ;ERROR E
U 0A4A, BE8B,15 ;9698          MOV WR[2] TO LS[ERROR.MASK]     ;SET UP TO CHECK BITS 30 AND 31
U 0A4B, B67E,95 ;9699          MOV LS[RDS] TO WR[1]            ;EXPECTED DATA (RDS SET, CRD CLEAR)
U 0A4C, 9D45,75 ;9700          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;9701                                         ;SET UP TO READ CSR 1
U 0A4D, 3022,15 ;9702          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0A4E, 0869,3C ;9703          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0A4F, 88A4,14 ;9704          JMP [LOOP.TE.D]                 ;ERROR LOOP IF ENABLED
                ;9705
                ;9706  TEST.TE.F:
U 0A50, B70C,15 ;9707          MOV LS[CKBTS.0111100] TO WR[0]  ;SET UP CKBTS FOR DATA OF ALL 0'S OR
                ;9708                                         ; ALL 1'S
U 0A51, A0C0,15 ;9709          COM WR[0]                       ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 0A52, 452C,15 ;9710          BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE
U 0A53, 4774,15 ;9711          BIS LS[DIAG.CHK] TO WR[0]       ;SET DIAG CHK BIT IN WR 0
U 0A54, 4778,15 ;9712          BIS LS[INH.CRD] TO WR[0]        ;SET INHIBIT REPORT CRD BIT
U 0A55, 8A17,FC ;9713          JSR [WRITE.CSR1]                ;WRITE CSR 1 WITH DATA FROM WR 0
                ;9714
U 0A56, FF82,15 ;9715          INC LS[ERROR.NUMBER]            ;ERROR F
U 0A57, 3683,95 ;9716          MOV LS[ERROR.NUMBER] TO WR[3]   ;SAVE ERROR NUMBER IN WR 3
```

```
                    ;9717   LOOP.TE.F:
U 0A58, BE83,95     ;9718           MOV WR[3] TO LS[ERROR.NUMBER]    ;SET UP ERROR NUMBER
U 0A59, 2F82,95     ;9719           CLR WR[1]                        ;EXPECTED DATA
U 0A5A, 9D9C,F5     ;9720           MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                    ;9721                                            ;SET UP TO WRITE CKBTS FROM CSR 1, DATA
                    ;9722                                            ; FROM LS
U 0A5B, 3240,15     ;9723           WRITE.MEM LS[#1]                 ;WRITE DATA OF 1 IN ECC CHIP (SINGLE BIT
                    ;9724                                            ; ERROR)
U 0A5C, 3022,15     ;9725           MOV MEM.DATA TO WR[0]            ;READ DATA BACK TO COMPLETE CYCLE (DON'T
                    ;9726                                            ; CHECK
U 0A5D, 9D45,75     ;9727           MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;9728                                            ;SET UP TO READ CSR 1
U 0A5E, 3022,15     ;9729           MOV MEM.DATA TO WR[0]            ;GET RESULT
U 0A5F, 0869,3C     ;9730           JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 0A60, 08A5,84     ;9731           JMP [LOOP.TE.F]                  ;ERROR LOOP IF ENABLED
                    ;9732
                    ;9733   TEST.TE.10:
U 0A61, FF82,15     ;9734           INC LS[ERROR.NUMBER]             ;ERROR 10
                    ;9735   LOOP.TE.10:
U 0A62, 5800,1D     ;9736           SKIP.IF[MEM.REF.OK]              ;SKIP IF NO MEMORY ERROR (SHOULD SKIP)
U 0A63, 08A7,DC     ;9737           JSR [ERROR.TE.10]                ;ERROR IF HERE. GO REPORT IT
                    ;9738
                    ;9739   TEST.TE.11:
U 0A64, FF82,15     ;9740           INC LS[ERROR.NUMBER]             ;ERROR 11
U 0A65, 3683,95     ;9741           MOV LS[ERROR.NUMBER] TO WR[3]    ;SAVE ERROR NUMBER IN WR 3
                    ;9742   LOOP.TE.11:
U 0A66, BE83,95     ;9743           MOV WR[3] TO LS[ERROR.NUMBER]    ;SET UP ERROR NUMBER
U 0A67, B67E,95     ;9744           MOV LS[RDS] TO WR[1]             ;EXPECTED DATA
U 0A68, 9D9C,F5     ;9745           MEM.REQ[MAINT.ECC.DATA] ADRS[#0] DT[LONG]
                    ;9746                                            ;SET UP TO WRITE CKBTS FROM CSR 1, DATA
                    ;9747                                            ; FROM LS
U 0A69, B2C0,15     ;9748           WRITE.MEM LS[#3(H)]              ;WRITE DATA OF 3(H) IN ECC CHIP (DOUBLE
                    ;9749                                            ; ERROR)
U 0A6A, 3022,15     ;9750           MOV MEM.DATA TO WR[0]            ;READ DATA BACK TO COMPLETE CYCLE (DON'T
                    ;9751                                            ; CHECK
U 0A6B, 9D45,75     ;9752           MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;9753                                            ;SET UP TO READ CSR 1
U 0A6C, 3022,15     ;9754           MOV MEM.DATA TO WR[0]            ;GET RESULT
U 0A6D, 0869,3C     ;9755           JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 0A6E, 88A6,64     ;9756           JMP [LOOP.TE.11]                 ;ERROR LOOP IF ENABLED
                    ;9757
                    ;9758   TEST.TE.12:
U 0A6F, FF82,15     ;9759           INC LS[ERROR.NUMBER]             ;ERROR 12
U 0A70, 5B00,1D     ;9760           SKIP.IF[MEM.REF.OK]              ;SKIP IF NO MEMORY ERROR (SHOULD NOT
                    ;9761                                            ; SKIP)
U 0A71, 88A8,D4     ;9762           JMP [END.TE]                     ;NO ERROR.  DONE WITH TEST
U 0A72, 88A8,0C     ;9763           JSR [ERROR.TE.12]                ;ERROR IF HERE. GO REPORT IT
                    ;9764
U 0A73, 88A8,D4     ;9765           JMP [END.TE]                     ;DONE WITH TEST
                    ;9766
                    ;9767
                    ;9768   ERROR.TE.4:
U 0A74, 88A8,3C     ;9769           JSR [REPORT.TE.SKIP.ERROR]       ;FORCE ERROR REPORT WITH EXP & REC N/A
U 0A75, 889C,F4     ;9770           JMP [LOOP.TE.1]                  ;ERROR LOOP IF ENABLED
U 0A76, 5B00,14     ;9771           RETURN                           ;NO ERROR LOOP RETURN
```

```
                    ;9772
                    ;9773   ERROR.TE.8:
U 0A77, 88A8,3C     ;9774           JSR [REPORT.TE.SKIP.ERROR]      ;FORCE ERROR REPORT WITH EXP & REC N/A
U 0A78, 08A1,04     ;9775           JMP [LOOP.TE.8]                 ;ERROR LOOP IF ENABLED
U 0A79, 5B00,14     ;9776           RETURN                          ;NO ERROR LOOP RETURN
                    ;9777
                    ;9778   ERROR.TE.C:
U 0A7A, 88A8,3C     ;9779           JSR [REPORT.TE.SKIP.ERROR]      ;FORCE ERROR REPORT WITH EXP & REC N/A
U 0A7B, 08A2,A4     ;9780           JMP [LOOP.TE.A]                 ;ERROR LOOP IF ENABLED
U 0A7C, 5B00,14     ;9781           RETURN                          ;NO ERROR LOOP RETURN
                    ;9782
                    ;9783   ERROR.TE.10:
U 0A7D, 88A8,3C     ;9784           JSR [REPORT.TE.SKIP.ERROR]      ;FORCE ERROR REPORT WITH EXP & REC N/A
U 0A7E, 08A5,84     ;9785           JMP [LOOP.TE.F]                 ;ERROR LOOP IF ENABLED
U 0A7F, 5B00,14     ;9786           RETURN                          ;NO ERROR LOOP RETURN
                    ;9787
                    ;9788   ERROR.TE.12:
U 0A80, 88A8,3C     ;9789           JSR [REPORT.TE.SKIP.ERROR]      ;FORCE ERROR REPORT WITH EXP & REC N/A
U 0A81, 88A6,64     ;9790           JMP [LOOP.TE.11]                ;ERROR LOOP IF ENABLED
U 0A82, 5B00,14     ;9791           RETURN                          ;NO ERROR LOOP RETURN
                    ;9792
                    ;9793   REPORT.TE.SKIP.ERROR:
U 0A83, B66E,15     ;9794           MOV LS[NA.EXP.REC] TO WR[0]     ;SET BIT 23 IN WR 0 TO INDICATE NOT TO
                    ;9795                                           ; PRINT EXPECTED OR RECEIVED DATA IF AN
                    ;9796                                           ; ERROR OCCURS
U 0A84, 3E80,15     ;9797           MOV WR[0] TO LS[CONTROL.STATUS] ;UPDATE CONTROL AND STATUS WORD
U 0A85, 2F80,15     ;9798           CLR WR[0]                       ;CLEAR WR 0
U 0A86, 369E,95     ;9799           MOV LS[ONES] TO WR[1]           ;SET WR 1.  NOW ERROR ROUTINE WILL
                    ;9800                                           ; REPORT AN ERROR (BUT EXP AND REC WILL
                    ;9801                                           ; NOT BE PRINTED)
U 0A87, 0869,3C     ;9802           JSR [CHECK.RESULT]              ;REPORT THE ERROR
U 0A88, 88A8,B4     ;9803           JMP [ERROR.LOOP.TE]             ;ERROR LOOP IF HERE
U 0A89, E580,15     ;9804           CLR LS[CONTROL.STATUS]          ;REENABLE EXP AND REC PRINTOUT
U 0A8A, DB00,16     ;9805           RETURN+1                        ;NO ERROR LOOP RETURN
                    ;9806   ERROR.LOOP.TE:
U 0A8B, E580,15     ;9807           CLR LS[CONTROL.STATUS]          ;REENABLE EXP AND REC PRINTOUT
U 0A8C, 5B00,14     ;9808           RETURN                          ;ERROR LOOP RETURN
                    ;9809
                    ;9810
                    ;9811   END.TE:
```

```
;9812    .PAGE   " Data Rotation Tests"
;9813
;9814    .toc    "  TEST F Rotate 9 Bits Right (M8391  MCT Module)"
;9815    ;++
;9816    ;
;9817    ; Test Description:
;9818    ;
;9819    ;       This test checks the data rotator PALS for a byte rotate right.  It
;9820    ;       issues a MEM.REQ with the MF field set to ROTATE.BYTE.RIGHT (6).  The
;9821    ;       data path is as follows:  A MEM.REQ is issued with the data pattern to
;9822    ;       be used coming from LS and the MF field=ROTATE.BYTE.RIGHT.  The data
;9823    ;       pattern is loaded into the VAR and a dispatch is made to the rotate
;9824    ;       right memory u-code (see dispatch description at the beginning of this
;9825    ;       listing).  At the dispatch address, memory busy is set, the PA bits
;9826    ;       are enabled, and VAR BYPASS is enabled.  In the next cycle, the data
;9827    ;       rotators are disabled, VAR BYPASS, ADDR PH, and memory busy are
;9828    ;       left asserted, and the MCT transceivers are enabled.  This puts the
;9829    ;       data pattern sent by the CPU back on the MC bus, shifted 1 bit to
;9830    ;       the right (due to the transceivers).  The next cycle will put the MC
;9831    ;       BUS bits onto the ARY BUS through the rotator PALS and open the ECC
;9832    ;       input latches so that the data can be saved.  The output latches of
;9833    ;       the ECC chip will also be enabled.  VAR BYPASS, ADDR PH, and memory
;9834    ;       busy are left asserted.  In the next cycle, the data stored in the ECC
;9835    ;       chip is enabled onto the ARY BUS and the rotator PAL is enabled to
;9836    ;       rotate this data 8 bits right and put the result on the MC BUS.  VAR
;9837    ;       BYPASS and memory busy are left asserted (there is no real reason that
;9838    ;       VAR BYPASS is left on, but it doesn't hurt anything).  The next cycle
;9839    ;       loops waiting for CPU DATA RCVD with memory busy cleared.  The MOV
;9840    ;       instruction in the CPU u-code will cause CPU DATA RCVD to be asserted
;9841    ;       and take the shifted data off the MC BUS for checking.  when CPU DATA
;9842    ;       RCVD is asserted, the MCT u-code will go to a location which jumps to
;9843    ;       the idle loop.  Data patterns used are shifting 1's.
;9844    ;
;9845    ; Assumptions:
;9846    ;
;9847    ;       It is assumed that the previous tests have run successfully.
;9848    ;
;9849    ; Test Steps:
;9850    ;
;9851    ;       1)  Set up error mask, error number, and module number in LS (for
;9852    ;           error printouts) and clear previous error number in LS.
;9853    ;       2)  Load WR 2 with data pattern from LS.
;9854    ;       3)  Put a 1 in bit 23 of WR 2 to get expected data in the right position.
;9855    ;           Store this in WR 1 as expected data.
;9856    ;       4)  Issue a MEM.REQ with DT=longword (11) and MF=ROTATE.BYTE.RIGHT (6)
;9857    ;           using the location in LS indexed by OS as the source of data.
;9858    ;       5)  Execute a MOV MEM.DATA to WR[0] and check the result.
;9859    ;       6)  Shift WR 2 left and repeat steps 4 and 5 until all data patterns
;9860    ;           have been tested.
;9861    ;
;9862    ; Errors:
;9863    ;
;9864    ;       Error 1 - Data rotation failed with data of shifting 1's.
;9865    ;
;9866    ; Debug:
```

```
                    ;9867   ;
                    ;9868   ;       Two methods of debug will be explained.  If the MCT module is in a
                    ;9869   ;       test station then the MCT can be single stepped making debug easier
                    ;9870   ;       in some cases.  Otherwise only the CPU can be single stepped and this
                    ;9871   ;       method will be explained also.  When single stepping the memory con-
                    ;9872   ;       troller will help in debug, this section will explicitly suggest it.
                    ;9873   ;
                    ;9874   ;       Error 1 - This error indicates a problem with the data rotator PALs
                    ;9875   ;       or the DATA ROTATOR CONTROL PAL. The inputs A0 L and A1 L at the data
                    ;9876   ;       rotator PAL can be checked after single stepping the CPU to the MOV
                    ;9877   ;       MEM.DATA to WR[0] instruction.  The memory will be looping waiting for
                    ;9878   ;       CPU DATA RCVD.  Check that A0 L is low and A1 L is high.  If these are
                    ;9879   ;       OK, check that 2ND MEM CYC H and MDR DAT OUT EN L are low.  If these
                    ;9880   ;       OK, suspect the data rotator PAL itself.  If A1 L or A0 L is wrong,
                    ;9881   ;       check them at the output of the DATA ROTATOR CONTROL PAL.  If they are
                    ;9882   ;       incorrect there and MCT single step is available, enable MCT single
                    ;9883   ;       step, step the CPU to the MEM.REQ instruction, and step the memory to
                    ;9884   ;       the cycle that executes SET.ROT.BYTE and ROT.CLOCK/CLOCK.  Check the
                    ;9885   ;       inputs to the DATA ROTATOR CONTROL PAL for a high on CPH/UBL and ROT
                    ;9886   ;       C0 H, and a low on ROT C1 H.  The signal ROT CLOCK H should also be
                    ;9887   ;       high.  If these are OK, the ROTATOR CONTROL PAL is probably bad.  If
                    ;9888   ;       MCT single step is not available, loop on error to check these inputs.
                    ;9889   ;         Another possible error is with the MDR AND DIR CONTROL PAL.  Cneck
                    ;9890   ;       the outputs that enable LAT OUTPUT BYT L to go high for lows (pins
                    ;9891   ;       12-15).  If incorrect check the inputs for a low on 2nd MEM CYC H,
                    ;9892   ;       and highs on CPH/UBL, ROT C0 H, A1 L and A0 L (during the time that the
                    ;9893   ;       ECC latches are open).  If these are OK, but pins 12-15 are not low,
                    ;9894   ;       the PAL is bad.
                    ;9895   ;--
                    ;9896
                    ;9897   T.F:
                    ;9898
U 0A8D, B65E,15     ;9899           MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;9900                                           ; STATUS WORD
U 0A8E, 3E80,15     ;9901           MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;9902                                           ; BIT 15 INDICATES START OF TEST TO
                    ;9903                                           ; CONSOLE
U 0A8F, 10E0,15     ;9904           MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;9905   WAIT.TF.0:
U 0A90, 88A9,04     ;9906           JMP [WAIT.TF.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                    ;9907                                           ; RESPOND
                    ;9908
U 0A91, 0A1A,AC     ;9909           JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;9910
U 0A92, E5F8,15     ;9911           CLR LS[OS]                      ;CLEAR INDEX
U 0A93, 366F,15     ;9912           MOV LS[BIT23] TO WR[2]          ;FIRST EXPECTED DATA PATTERN
                    ;9913
                    ;9914   LOOP.TF.1:
U 0A94, A004,95     ;9915           MOV WR[2] TO WR[1]              ;EXPECTED DATA
U 0A95, 19F7,75     ;9916           MEM.REQ[ROTATE.BYTE.RIGHT] ADRS[SHIFT.OS(4-0)] DT[LONG]
                    ;9917                                           ;DO SHIFT RIGHT ON DATA
U 0A96, 3022,15     ;9918           MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0A97, 0869,3C     ;9919           JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0A98, 08A9,44     ;9920           JMP [LOOP.TF.1]                 ;ERROR LOOP IF ENABLED
                    ;9921
```

```
U 0A99  23C1,15 ;9922            ROL WR[2]                          ;NEXT EXPECTED DATA
U 0A9A, 7FF8,15 ;9923            INC LS[OS]                         ;INCREMENT INDEX FOR NEXT PATTERN
U 0A9B, B6F8,15 ;9924            MOV LS[OS] TO WR[0]                ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                ;9925            BIT LS[BIT5] WITH WR[0],           ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0A9C, 594A,35 ;9926             DT(LONG)&SET.ALU.CC               ; AND SET CONDITION CODES
U 0A9D, 88A9,49 ;9927            JMP.IF[BITS.CLR] TO [LOOP.TF.1] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                ;9928   END.TF:
```

```
;9929   .PAGE   "  TEST 10 Rotate 15 Bits Left (M8391  MCT Module)"
;9930   ;++
;9931   ;
;9932   ; Test Description:
;9933   ;
;9934   ;       This test checks the data rotator PALS for a word rotate left.  It
;9935   ;       issues a MEM.REQ with the MF field set to WORD.SWAP (7).  The data
;9936   ;       path is as follows:  A MEM.REQ is issued with the data pattern to
;9937   ;       be used coming from LS and the MF field=WORD.SWAP.  The data
;9938   ;       pattern is loaded into the VAR and a dispatch is made to the rotate
;9939   ;       left memory u-code (see dispatch description at the beginning of this
;9940   ;       listing).  At the dispatch address, memory busy is set, the PA bits
;9941   ;       are enabled, and VAR BYPASS is enabled.  In the next cycle, the data
;9942   ;       rotators are disabled, VAR BYPASS, ADDR PH, and memory busy are
;9943   ;       left asserted, and the MCT transceivers are enabled.  This puts the
;9944   ;       data pattern sent by the CPU back on the MC bus, shifted 1 bit to
;9945   ;       the right (due to the transceivers).  The next cycle will put the MC
;9946   ;       BUS bits onto the ARY BUS through the rotator PALS and open the ECC
;9947   ;       input latches so that the data can be saved.  The output latches of
;9948   ;       the ECC chip will also be enabled.  VAR BYPASS, ADDR PH, and memory
;9949   ;       busy are left asserted.  In the next cycle, the data stored in the ECC
;9950   ;       chip is enabled onto the ARY BUS and the rotator PAL is enabled to
;9951   ;       rotate this data 16 bits left and put the result on the MC BUS.  VAR
;9952   ;       BYPASS and memory busy are left asserted (there is no real reason that
;9953   ;       VAR BYPASS is left on, but it doesn't hurt anything).  The next cycle
;9954   ;       loops waiting for CPU DATA RCVD with memory busy cleared.  The MOV
;9955   ;       instruction in the CPU u-code will cause CPU DATA RCVD to be asserted
;9956   ;       and take the shifted data off the MC BUS for checking.  when CPU DATA
;9957   ;       RCVD is asserted, the MCT u-code will go to a location which jumps to
;9958   ;       the idle loop.  Data patterns used are shifting 1's.
;9959   ;
;9960   ; Assumptions:
;9961   ;
;9962   ;       It is assumed that the previous tests have run successfully.
;9963   ;
;9964   ; Test Steps:
;9965   ;
;9966   ;       1)  Set up error mask, error number, and module number in LS (for
;9967   ;           error printouts) and clear previous error number in LS.
;9968   ;       2)  Load WR 2 with data pattern from LS.
;9969   ;       3)  Put a 1 in bit 15 of WR 2 to get expected data in the right position.
;9970   ;           Store this in WR 1 as expected data.
;9971   ;       4)  Issue a MEM.REQ with DT=longword (11) and MF=WORD.SWAP (7) using
;9972   ;           the location in LS indexed by OS as the source of data.
;9973   ;       5)  Execute a MOV MEM.DATA to WR[0] and check the result.
;9974   ;       6)  Shift WR 2 left and repeat steps 4 and 5 until all data patterns
;9975   ;           have been tested.
;9976   ;
;9977   ; Errors:
;9978   ;
;9979   ;       Error 1 - Data rotation failed with data of shifting 1's.
;9980   ;
;9981   ; Debug:
;9982   ;
;9983   ;       Two methods of debug will be explained.  If the MCT module is in a
```

```
                    ;9984   ;       test station then the MCT can be single stepped making debug easier
                    ;9985   ;       in some cases.  Otherwise only the CPU can be single stepped and this
                    ;9986   ;       method will be explained also.  When single stepping the memory con-
                    ;9987   ;       troller will help in debug, this section will explicitly suggest it.
                    ;9988   ;
                    ;9989   ;       Error 1 - This error indicates a problem with the data rotator PALs
                    ;9990   ;       or the DATA ROTATOR CONTROL PAL. The inputs A0 L and A1 L at the data
                    ;9991   ;       rotator PAL can be checked after single stepping the CPU to the MOV
                    ;9992   ;       MEM.DATA to WR[0] instruction.  The memory will be looping waiting for
                    ;9993   ;       CPU DATA RCVD.  Check that A0 L is high and A1 L is low.  If these are
                    ;9994   ;       OK, check that 2ND MEM CYC H and MDR DAT OUT EN L are low.  If these
                    ;9995   ;       OK, suspect the data rotator PAL itself.  If A1 L or A0 L is wrong,
                    ;9996   ;       check them at the output of the DATA ROTATOR CONTROL PAL.  If they are
                    ;9997   ;       incorrect there and MCT single step is available, enable MCT single
                    ;9998   ;       step, step the CPU to the MEM.REQ instruction, and step the memory to
                    ;9999   ;       the cycle that executes SET.ROT.BYTE and ROT.CLOCK/CLOCK.  Check the
                    ;10000  ;       inputs to the DATA ROTATOR CONTROL PAL for a high on CPH/UBL and ROT
                    ;10001  ;       C1 H, and a low on ROT C0 H.  The signal ROT CLOCK H should also be
                    ;10002  ;       high.  If these are OK, the ROTATOR CONTROL PAL is probably bad.  If
                    ;10003  ;       MCT single step is not available, loop on error to check these inputs.
                    ;10004  ;--
                    ;10005
                    ;10006  T.10:
                    ;10007
U 0A9E, B65E,15     ;10008          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;10009                                          ; STATUS WORD
U 0A9F, 3E80,15     ;10010          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;10011                                          ; BIT 15 INDICATES START OF TEST TO
                    ;10012                                          ; CONSOLE
U 0AA0, 10E0,15     ;10013          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;10014  WAIT.T10.0:
U 0AA1, 08AA,14     ;10015          JMP [WAIT.T10.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;10016                                          ; RESPOND
                    ;10017
U 0AA2, 0A1A,AC     ;10018          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;10019
U 0AA3, E5F8,15     ;10020          CLR LS[OS]                      ;CLEAR INDEX
U 0AA4, 365F,15     ;10021          MOV LS[BIT15] TO WR[2]          ;FIRST EXPECTED DATA PATTERN
                    ;10022  LOOP.T10.1:
U 0AA5, A004,95     ;10023          MOV WR[2] TO WR[1]              ;EXPECTED DATA
U 0AA6, 99F7,F5     ;10024          MEM.REQ[WORD.SWAP] ADRS[SHIFT.OS(4-0)] DT[LONG]
                    ;10025                                          ;DO SHIFT LEFT ON DATA
U 0AA7, 3022,15     ;10026          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0AA8, 0869,3C     ;10027          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0AA9, 88AA,54     ;10028          JMP [LOOP.T10.1]                ;ERROR LOOP IF ENABLED
                    ;10029
U 0AAA, 23C1,15     ;10030          ROL WR[2]                       ;NEXT EXPECTED DATA PATTERN
U 0AAB, 7FF8,15     ;10031          INC LS[OS]                      ;INCREMENT INDEX FOR NEXT PATTERN
U 0AAC, B6F8,15     ;10032          MOV LS[OS] TO WR[0]             ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                    ;10033          BIT LS[BIT5] WITH WR[0],        ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0AAD, 594A,35     ;10034           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0AAE, 08AA,59     ;10035          JMP.IF[BITS.CLR] TO [LOOP.T10.1] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                    ;10036  END.T10:
```

```
;10037  .PAGE   " Translation Buffer, Unibus Map, and TAG Tests"
;10038
;10039  .toc    "  TEST 11 TB RAM Test (M8391  MCT Module)"
;10040  ;++
;10041  ;
;10042  ; Test Description:
;10043  ;
;10044  ;       This test checks the CPU portion of the TB (translation buffer) RAMS.
;10045  ;       Addresses 0 to 7F(H) can be accessed (128 decimal addresses) using the
;10046  ;       WRITE.TB memory controller micro-routine.  The same addresses can be
;10047  ;       read back for verification using the READ.TB routine.  Other addresses
;10048  ;       in the 1K by 4 RAM are either unused or tested in the UNIBUS MAP test.
;10049  ;       The address accessed in the TB RAM is determined by VAR bits 09 through
;10050  ;       14 and VAR bit 31.  RAM address bits 7 through 9 are forced to 0 by the
;10051  ;       hardware.  VAR bits 0 through 8 and 15 through 30 have no effect on the
;10052  ;       TB RAM address selected.  The address is sent from LS during a MEM.REQ
;10053  ;       CPU instruction with the MF field = WRITE.TB (1H) or READ.TB (13H).  A
;10054  ;       WRITE.MEM LS CPU instruction sends the data to be written in the TB in
;10055  ;       the following manner:  LS bits 0-15 go to PA bits 09-24 respectively,
;10056  ;       and LS bits 25-31 go to the byte offset, modify, prot A, prot B, prot
;10057  ;       C, prot D, and valid bits respectively.  The data to write in the TB is
;10058  ;       rotated in the data latches and stored in the ECC latches.  It is then
;10059  ;       read out of the latches and written into the TB.  Note that PA 24 is
;10060  ;       not used, only PA 9-23 is written in the TB.
;10061  ;         The data path for the write is as follows:  A MEM.REQ is issued with
;10062  ;       MF=WRITE.TB and DT=longword.  The dispatch  occurs as described in the
;10063  ;       beginning of this listing.  At the dispatch address, DATI is set to
;10064  ;       clear ICO, the transceivers are set up to receive data from the CPU
;10065  ;       to write into the TB, and the transcievers on the MCT are set up to
;10066  ;       pass data from the MC bus to the PA bits and the other TB data bits.
;10067  ;       Memory busy is set and rotate byte is set.  The next cycle repeats this
;10068  ;       sequence and adds a ROT CLK to enable the rotate byte function.  The
;10069  ;       following cyc   loops waiting for CPU DATA REQ (which comes up on the
;10070  ;       WRITE.MEM LS CPU instruction).  It drops memory busy to release the CPU
;10071  ;       stall, enables the CPU transceivers to receive the data from the CPU,
;10072  ;       and sets up the MCT transceivers to pass data from MC BUS to TB.  When
;10073  ;       CPU DATA REQ becomes active, the next cycle enables the data in the
;10074  ;       rotators onto the ARRAY BUS and opens the ECC latches to receive this
;10075  ;       data.  The CPU transceivers are enabled to receive data from the CPU,
;10076  ;       the MCT transceivers are set up to pass data to the TB, and CPUG is
;10077  ;       cleared (this is to make sure CPU GRANT is dropped even if the CPU is
;10078  ;       single stepping).  The next cycle simply sets rotate disable and keeps
;10079  ;       the MCT transceivers set up to pass data to the TB.  It drops open ECC
;10080  ;       so the data is now latched in the ECC chip.  The following cycle issues
;10081  ;       a rotate clock to disable rotation and enables the ECC data onto the
;10082  ;       array bus.  The rotators are enabled to put this data on the MC BUS,
;10083  ;       and the MCT transcievers are enabled to put the MC BUS data onto the TB
;10084  ;       data inputs.  The write enable on the TB is enabled, but the write will
;10085  ;       not occur until write enable is removed (goes high).  The next cycle
;10086  ;       repeats this except for the rotate clk.  This cycle will branch on ICO
;10087  ;       which was cleared with the set DATI in the dispatch cycle.  The last
;10088  ;       keeps all the input data to the TB enabled but drops TB write enable.
;10089  ;       This will cause the write into TB to occur.  It then goes to the idle
;10090  ;       loop.
;10091  ;         The read flows are initiated by a MEM.REQ with MF=READ.TB (13H).  The
```

```
;10092  ;       dispatch occurs as described in the beginning of this listing.  At the
;10093  ;       dispatch address, VAR bypass is enabled, memory busy is set, and the
;10094  ;       MCT transceiver that passes the TB data other than the PA bits is en-
;10095  ;       abled.  The next cycle keeps memory busy, the MCT transceiver, and VAR
;10096  ;       bypass enabled, and enables the other MCT transciever that passes the
;10097  ;       PA bits.  Now that the TB data is on the MC BUS, the following cycle
;10098  ;       drops memory busy while keeping the MCT transceivers enabled, and loops
;10099  ;       waiting for CPU GRANT to drop (this occurs when the MOV MEM.DATA TO WR
;10100  ;       CPU instruction is executed and asserts CPU DATA RCVD).  When CPU GRANT
;10101  ;       drops, the last cycle prepares for a UNIBUS cycle and goes to idle.
;10102  ;       The TB address was supplied during the MEM.REQ instruction which start-
;10103  ;       this TB read flow.  The data comes back unshifted.  Therefore byte off-
;10104  ;       set through valid comes back on bits 01 through 07 respectively,PA 09-
;10105  ;       23 comes back on bits 08-22 respectively.
;10106  ;         The above cycles will be used with data patterns of all 1's, all 0's,
;10107  ;       shifting 1,s and shifting 0's.  Then a shifting address bit test will
;10108  ;       be run to check address lines.  Finally a moving inversion test will be
;10109  ;       executed to check for slow address logic.  This test will run like a
;10110  ;       march test, except it is run 7 times, each time toggeling a different
;10111  ;       address bit at the maximum rate.  The march will write a background of
;10112  ;       zeros, then increment through the TB doing a read 0's, write 1's,
;10113  ;       sequence.  When the last address has been checked, the march will
;10114  ;       decrement through the TB executing a read 1's, write 0's, sequence.
;10115  ;
;10116  ;
;10117  ; Assumptions:
;10118  ;
;10119  ;       It is assumed that all previous tests have run successfully.
;10120  ;
;10121  ; Test Steps:
;10122  ;
;10123  ;       1)  Set up error mask, error number, and module number in LS (for
;10124  ;           error printouts) and clear previous error number in LS.  The error
;10125  ;           mask will allow bits 1 through 22 to be checked.
;10126  ;       2)  Load WR 1 with all 1's, and execute a MEM.REQ with MF=WRITE.TB and
;10127  ;           DT=longword.  Use an LS location containing 0's as the address.
;10128  ;       3)  Execute a WRITE.MEM LS with an LS location containing all 1's.
;10129  ;       4)  Execute a MEM.REQ with MF=READ.TB and DT=longword.  Use an LS loc-
;10130  ;           ation containing 0's as the address.
;10131  ;       5)  Execute a MOV MEM.DATA to WR[0] and check result for 1's.
;10132  ;       6)  Repeat steps 2 through 5 with data of 0's in step 3 and 5.
;10133  ;       7)  Repeat steps 2 through 5 with data of shifting 1's in step 3 and 5.
;10134  ;           Expected data in WR 1 is data sent shifted 8 bits left.
;10135  ;       8)  Repeat steps 2 through 5 with data of shifting 0's in step 3 and 5.
;10136  ;           Expected data in WR 1 is data sent shifted 8 bits left.
;10137  ;       9)  Repeat steps 2 through 5 with data patterns of 0's, 1's, shifting
;10138  ;           1's and shifting 0's, incrementing through all addresses with each
;10139  ;           data pattern.  This verifies all RAM cells that can be accessed.
;10140  ;           Expected data in WR 1 is data sent shifted 8 bits left.
;10141  ;       10) Execute a march type pattern while changing the address by shifting
;10142  ;           a 1 bit across an address field of zeros.  Write a background of
;10143  ;           0's, then read 0's, write 1's, read 0's.  This will be repeated 7
;10144  ;           times while shifting a 1 across the address lines.  This will ver-
;10145  ;           ify the address lines and gross failures in the RAM chip.
;10146  ;       11) Execute a moving inversion test (described in description above).
```

```
;10147  ;
;10148  ;
;10149  ; Errors:
;10150  ;
;10151  ;       NOTE:  OTHER data is TB address being tested.
;10152  ;
;10153  ;       Error 1 - Write or read TB failed with data of all ones, address 0.
;10154  ;       Error 2 - Write or read TB failed with data of all zeros, address 0.
;10155  ;       Error 3 - Write or read TB failed with data of shifting 1's, address 0.
;10156  ;       Error 4 - Write or read TB failed with data of shifting 0's, address 0.
;10157  ;       Error 5 - TB failure with data of all ones.
;10158  ;       Error 6 - TB failure with data of all zeros
;10159  ;       Error 7 - TB failure with data of shifting ones.
;10160  ;       Error 8 - TB failure with data of shifting zeros.
;10161  ;       Error 9 - TB address failure, shifting address bit test.
;10162  ;       Error A - TB failure with moving inversion pattern,ascending addresses.
;10163  ;       Error B - TB failure with moving inversion pattern,descending addresses.
;10164  ;
;10165  ; Debug:
;10166  ;
;10167  ;       ***NOTE***The TB address that failed will be printed under "other data"
;10168  ;       in the error report (except for errors A and D).  The bits will be
;10169  ;       arranged so as to give the actual RAM address in hexadecimal format.
;10170  ;
;10171  ;       Two methods of debug will be explained.  If the MCT module is in a
;10172  ;       test station then the MCT can be single stepped making debug easier
;10173  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;10174  ;       method will be explained also.  When single stepping the memory con-
;10175  ;       troller will help in debug, this section will explicitly suggest it.
;10176  ;
;10177  ;       Error 1 - This error indicates a basic problem with the TB logic.  If
;10178  ;       MCT single step is available, step the CPU to the MEM.REQ instruction
;10179  ;       and enable MCT single step.  Step the CPU to the WRITE.MEM LS instruc-
;10180  ;       tion.  Now step the MCT to the cycle that branches on ICO (BEN 2=
;10181  ;       LUB.CO).  Check the inputs BUS MC D01-BUS MC D23 going into the trans-
;10182  ;       ceivers that output the PA bits and byte offset through valid.  The BUS
;10183  ;       MC Dxx lines should all be high.  If not, one of the DATA ROTATOR PALS
;10184  ;       is probably bad (the other circuitry up to here has already been test-
;10185  ;       ed).  The defective PAL is the one supplying the bad BUS MC signal.
;10186  ;         If the BUS MC signals are OK, check the signal TB DATA DIR RD L for
;10187  ;       a high, TB DATA EN L for a low, and TB DATA EN+MAINT L for a low into
;10188  ;       all three latches.  These come directly from the control store.  If
;10189  ;       these are OK, the output of the latches (the B side) should be all
;10190  ;       high.  If not, suspect the latch outputing the defective bit.
;10191  ;         If the output of the latches is OK, check the inputs into the 1KX4
;10192  ;       RAM for all highs.  An error here indicates a bad etch run.  Check the
;10193  ;       input TB WE L for a low on pin 10 of each of the 1KX4 RAM chips.  This
;10194  ;       signal comes directly from the control store.  If this signal is OK,
;10195  ;       the RAM should have the data written on the next cycle when TB WE L
;10196  ;       goes high.
;10197  ;         Single step the MCT one more cycle and check that TB WE L goes high,
;10198  ;       that ADDR PH H is low, and that the outputs on pins 11-14 of the RAM
;10199  ;       chips are high.  If the inputs are good, but an output is low, the RAM
;10200  ;       chip is probably bad.
;10201  ;         If everything is OK up to now, then the READ.TB routine is failing.
```

```
                ;10202  ;       Single step the CPU to the MOV MEM.DATA TO WR[0] instruction.  The MCT
                ;10203  ;       will be looping waiting for CPU GRANT to drop.  Check the signal TB
                ;10204  ;       DATA DIR RD L and TB DATA EN L for a low into the MCT transceivers.
                ;10205  ;       Also check that the inputs on side B are all high.  If these signals
                ;10206  ;       are OK, then check the outputs on side A for nighs.  A problem here in-
                ;10207  ;       dicates a bad transceiver.
                ;10208  ;
                ;10209  ;       Error 2 - Same as error 1 except that the data inputs to the trans-
                ;10210  ;       ceivers and RAM chips are low instead of high.
                ;10211  ;
                ;10212  ;       Error 3 - This error indicates a possible short between data lines go-
                ;10213  ;       ing to the TB or in the RAM chip itself.
                ;10214  ;
                ;10215  ;       Error 4 - Same as error 3.
                ;10216  ;
                ;10217  ;       Error 5 through 8 - This error most likely indicates a bad RAM chip at
                ;10218  ;       the address specified under "OTHER".
                ;10219  ;
                ;10220  ;       Error 9 - This error indicates a problem with either the address lines
                ;10221  ;       or an internal address problem within a chip.  If all bits are failing,
                ;10222  ;       suspect an address line.  The address lines can be examined by enabling
                ;10223  ;       SOMM mode at the MEM.REQ with MF=WRITE.TB.  Each time the CPU halts on
                ;10224  ;       the micro-match, the address lines can be examined on the MCT module.
                ;10225  ;       A one should be shifting across the address lines from LVA 09 H through
                ;10226  ;       LVA 14 H and then to TBA 6 H after a CONTINUE is issued.  TBA 7 through
                ;10227  ;       TBA 9 should always be low.  If TBA 6 H fails to go high, check the 4X2
                ;10228  ;       MUX that supplies the TBA address as outputs.  The select signal UB TB
                ;10229  ;       SEL H should be low.  This can be traced back to the negated input 'OR'
                ;10230  ;       gate with CPH/UBL H and UB TB SEL L as inputs.  Both these inputs
                ;10231  ;       should be high.  If not, suspect the PAL that generates them.
                ;10232  ;         Another possibility is that the address is failing on the READ.TB
                ;10233  ;       micro-flow.  This flow uses the VAR BYPASS REGISTER for the first time.
                ;10234  ;       Setting SOMM on the MEM.REQ with MF=READ.TB will allow the address
                ;10235  ;       lines to be examined.  After each CONTINUE the address lines should
                ;10236  ;       be seen to shift a one across a field of zeros.
                ;10237  ;         If all bits are not failing, suspect a bad cell in the RAM chip it-
                ;10238  ;       self.  If a group of four bits is failing, suspect a break in the ad-
                ;10239  ;       dress line etch to that RAM chip.
                ;10240  ;
                ;10241  ;       Error A and B - This error most likely indicates a bad RAM chip,
                ;10242  ;       either in a single cell or in the address switching logic.
                ;10243  ;         The address can be determined by examining LS location 9(H).  Bits
                ;10244  ;       9-14 correspond to bits 0-5 at the RAM chip.  Bit 31 corresponds to
                ;10245  ;       bit 6 at the RAM chip.
                ;10246  ;
                ;10247  ;--
                ;10248
                ;10249  T.11:
                ;10250
U 0AAF, B65E,15 ;10251          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;10252                                          ; STATUS WORD
U CAB0, 3E80,15 ;10253          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;10254                                          ; BIT 15 INDICATES START OF TEST TO
                ;10255                                          ; CONSOLE
U 0AB1, 10E0,15 ;10256          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
```

```
                    ;10257  WAIT.T11.0:
U 0AB2, 88AB,24     ;10258          JMP [WAIT.T11.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;10259                                          ; RESPOND
                    ;10260
U 0AB3, 0A1A,AC     ;10261          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;10262
U 0AB4, B62A,15     ;10263          MOV LS[#FF000000] TO WR[0]      ;SET TOP BYTE FOR ERROR MASK
U 0AB5, C76E,15     ;10264          BIS LS[BIT23] TO WR[0]          ;SET BIT 23
U 0AB6, C740,15     ;10265          BIS LS[BIT0] TO WR[0]           ;AND SET BIT 0.  NOW BITS 1 THROUGH 22
                    ;10266                                          ; WILL BE CHECKED FOR ERRORS
U 0AB7, 3E8A,15     ;10267          MOV WR[0] TO LS[ERROR.MASK]     ;STORE IN LS ERROR MASK
U 0AB8, 6588,15     ;10268          CLR LS[ADDRESS.DATA]            ;LOAD A 0 TO BE PRINTED UNDER 'OTHER'
                    ;10269                                          ; IN THE ERROR REPORT, AS THE ADDRESS
U 0AB9, B670,15     ;10270          MOV LS[OTHER.DATA] TO WR[0]     ;SET BIT 24 IN WR 0
U 0ABA, 3E80,15     ;10271          MOV WR[0] TO LS[CONTROL.STATUS] ;CONSOLE WILL PRINT UNDER 'OTHER' IF
                    ;10272                                          ; ERROR
                    ;10273
                    ;10274  LOOP.T11.1:
U 0ABB, 369E,95     ;10275          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 0ABC, 989C,F5     ;10276          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                    ;10277                                          ;SET UP TO WRITE TB ADDRESS 0
U 0ABD, 329E,15     ;10278          WRITE.MEM LS[ONES]              ;WRITE ALL ONES IN TB
U 0ABE, 9C9D,F5     ;10279          MEM.REQ[READ.TB] ADRS[#0] DT[LONG]
                    ;10280                                          ;SET UP TO READ TB
U 0ABF, 3022,15     ;10281          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0AC0, 0869,3C     ;10282          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0AC1, 88AB,B4     ;10283          JMP [LOOP.T11.1]                ;ERROR LOOP IF ENABLED
                    ;10284
U 0AC2, FF82,15     ;10285          INC LS[ERROR.NUMBER]            ;ERROR 2
                    ;10286  LOOP.T11.2:
U 0AC3, B69C,95     ;10287          MOV LS[ZERO] TO WR[1]           ;EXPECTED DATA
U 0AC4, 989C,F5     ;10288          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                    ;10289                                          ;SET UP TO WRITE TB ADDRESS 0
U 0AC5, B29C,15     ;10290          WRITE.MEM LS[ZERO]              ;WRITE ALL ZEROS IN TB
U 0AC6, 9C9D,F5     ;10291          MEM.REQ[READ.TB] ADRS[#0] DT[LONG]
                    ;10292                                          ;SET UP TO READ TB
U 0AC7, 3022,15     ;10293          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0AC8, 0869,3C     ;10294          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0AC9, 88AC,34     ;10295          JMP [LOOP.T11.2]                ;ERROR LOOP IF ENABLED
                    ;10296
U 0ACA, FF82,15     ;10297          INC LS[ERROR.NUMBER]            ;ERROR 3
U 0ACB, E5F8,15     ;10298          CLR LS[OS]                      ;CLEAR INDEX
                    ;10299
U 0ACC, 3651,95     ;10300          MOV LS[BIT8] TO WR[3]           ;FIRST EXPECTED DATA (RECEIVED DATA)
                    ;10301                                          ; IS SHIFTED LEFT 8 PLACES)
                    ;10302  LOOP.T11.3:
U 0ACD, 2006,95     ;10303          MOV WR[3] TO WR[1]              ;EXPECTED DATA
U 0ACE, 989C,F5     ;10304          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                    ;10305                                          ;SET UP TO WRITE TB ADDRESS 0
U 0ACF, B2F6,15     ;10306          WRITE.MEM LS[SHIFT.OS(4-0)]     ;WRITE TB WITH SHIFTING 1'S DATA
U 0AD0, 9C9D,F5     ;10307          MEM.REQ[READ.TB] ADRS[#0] DT[LONG]
                    ;10308                                          ;SET UP TO READ TB ADDRESS 0
U 0AD1, 3022,15     ;10309          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0AD2, 0869,3C     ;10310          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0AD3, 08AC,D4     ;10311          JMP [LOOP.T11.3]                ;ERROR LOOP IF ENABLED
```

```
                    ;10312
U 0AD4, A3C1,95     ;10313          ROL WR[3]                           ;NEXT EXPECTED DATA PATTERN
U 0AD5, 7FF8,15     ;10314          INC LS[OS]                          ;INCREMENT INDEX FOR NEXT PATTERN
U 0AD6, B6F8,15     ;10315          MOV LS[OS] TO WR[0]                 ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                    ;10316          BIT LS[BIT5] WITH WR[0],            ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0AD7, 594A,35     ;10317           DT(LONG)&SET.ALU.CC                ; AND SET CONDITION CODES
U 0AD8, 88AC,D9     ;10318          JMP.IF[BITS.CLR] TO [LOOP.T11.3] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                    ;10319
U 0AD9, FF82,15     ;10320          INC LS[ERROR.NUMBER]                ;ERROR 4
U 0ADA, E5F8,15     ;10321          CLR LS[OS]                          ;CLEAR INDEX
U 0ADB, 5F51,95     ;10322          MCOM LS[BIT8] TO WR[3]              ;FIRST EXPECTED DATA (RECEIVED DATA)
                    ;10323                                              ; IS SHIFTED LEFT 8 PLACES)
                    ;10324  REPEAT.T11.4:
U 0ADC, 5FF6,15     ;10325          MCOM LS[SHIFT.OS(4-0)] TO WR[0] ;GET CURRENT DATA PATTERN
U 0ADD, 3E10,15     ;10326          MOV WR[0] TO LS[T8]                 ;PUT DATA PATTERN IN TEMPORARY LS
                    ;10327  LOOP.T11.4:
U 0ADE, 2006,95     ;10328          MOV WR[3] TO WR[1]                  ;EXPECTED DATA
U 0ADF, 989C,F5     ;10329          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                    ;10330                                              ;SET UP TO WRITE TB ADDRESS 0
U 0AE0, 3210,15     ;10331          WRITE.MEM LS[T8]                    ;WRITE TB WITH SHIFTING 0'S DATA
U 0AE1, 9C9D,F5     ;10332          MEM.REQ[READ.TB] ADRS[#0] DT[LONG]
                    ;10333                                              ;SET UP TO READ TB ADDRESS 0
U 0AE2, 3022,15     ;10334          MOV MEM.DATA TO WR[0]               ;GET RESULT
U 0AE3, 0869,3C     ;10335          JSR [CHECK.RESULT]                  ;CHECK FOR ERRORS
U 0AE4, 88AD,E4     ;10336          JMP [LOOP.T11.4]                    ;ERROR LOOP IF ENABLED
                    ;10337
U 0AE5, A3C1,95     ;10338          ROL WR[3]                           ;NEXT EXPECTED DATA PATTERN
U 0AE6, 7FF8,15     ;10339          INC LS[OS]                          ;INCREMENT INDEX FOR NEXT PATTERN
U 0AE7, B6F8,15     ;10340          MOV LS[OS] TO WR[0]                 ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                    ;10341          BIT LS[BIT5] WITH WR[0],            ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0AE8, 594A,35     ;10342           DT(LONG)&SET.ALU.CC                ; AND SET CONDITION CODES
U 0AE9, 88AD,C9     ;10343          JMP.IF[BITS.CLR] TO [REPEAT.T11.4] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                    ;10344
U 0AEA, FF82,15     ;10345          INC LS[ERROR.NUMBER]                ;ERROR 5
U 0AEB, 6512,15     ;10346          CLR LS[T9]                          ;CLEAR TB ADDRESS (ADDRESS STORED IN LS 9)
U 0AEC, B640,15     ;10347          MOV LS[#1] TO WR[0]                 ;GET A 1 IN WR 0
U 0AED, BEFC,15     ;10348          MOV WR[0] TO LS[SIZE]               ;LOAD SIZE REG WITH CODE FOR DATA TYPE
                    ;10349                                              ; OF WORD
                    ;10350  LOOP.T11.5:
U 0AEE, 369E,95     ;10351          MOV LS[ONES] TO WR[1]               ;EXPECTED DATA
U 0AEF, 9812,F5     ;10352          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                    ;10353                                              ;SET UP TO WRITE TB
U 0AF0, 329E,15     ;10354          WRITE.MEM LS[ONES]                  ;WRITE ALL ONES IN TB
U 0AF1, 9C13,F5     ;10355          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                    ;10356                                              ;SET UP TO READ TB
U 0AF2, 3022,15     ;10357          MOV MEM.DATA TO WR[0]               ;GET RESULT
U 0AF3, 0869,3C     ;10358          JSR [CHECK.RESULT]                  ;CHECK FOR ERRORS
U 0AF4, 88AE,E4     ;10359          JMP [LOOP.T11.5]                    ;ERROR LOOP IF ENABLED
                    ;10360
U 0AF5, 0A18,2C     ;10361          JSR [NEXT.TB.ADDRESS]               ;CALCULATE NEXT TB ADDRESS
U 0AF6, 88AE,E4     ;10362          JMP [LOOP.T11.5]                    ;RETURN HERE IF NOT DONE ALL ADDRESSES
                    ;10363
U 0AF7, FF82,15     ;10364          INC LS[ERROR.NUMBER]                ;ERROR 6
U 0AF8, 6512,15     ;10365          CLR LS[T9]                          ;CLEAR TB ADDRESS (ADDRESS STORED IN LS 9)
U 0AF9, 6588,15     ;10366          CLR LS[ADDRESS.DATA]                ;CLEAR ADDRESS INFO TO BE PRINTED IF
```

```
                    ;10367                                          ; ERROR
                    ;10368  LOOP.T11.6:
U 0AFA, B69C.95     ;10369          MOV LS[ZERO] TO WR[1]            ;EXPECTED DATA
U 0AFB, 9812.F5     ;10370          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                    ;10371                                          ;SET UP TO WRITE TB
U 0AFC, B29C.15     ;10372          WRITE.MEM LS[ZERO]               ;WRITE ALL ZEROS IN TB
U 0AFD, 9C13.F5     ;10373          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                    ;10374                                          ;SET UP TO READ TB
U 0AFE, 3022.15     ;10375          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 0AFF, 0869.3C     ;10376          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 0B00, 88AF.A4     ;10377          JMP [LOOP.T11.6]                 ;ERROR LOOP IF ENABLED
                    ;10378
U 0B01, 0A18.2C     ;10379          JSR [NEXT.TB.ADDRESS]            ;CALCULATE NEXT TB ADDRESS
U 0B02, 88AF.A4     ;10380          JMP [LOOP.T11.6]                 ;RETURN HERE IF NOT DONE ALL ADDRESSES
                    ;10381
U 0B03, FF82.15     ;10382          INC LS[ERROR.NUMBER]             ;ERROR 7
U 0B04, E5F8.15     ;10383          CLR LS[OS]                       ;CLEAR INDEX
U 0B05, 6512.15     ;10384          CLR LS[T9]                       ;CLEAR TB ADDRESS (ADDRESS STORED IN LS 9)
                    ;10385
U 0B06, 3651.95     ;10386          MOV LS[BIT8] TO WR[3]            ;FIRST EXPECTED DATA (RECEIVED DATA)
U 0B07, 6588.15     ;10387          CLR LS[ADDRESS.DATA]             ;CLEAR ADDRESS INFO TO BE PRINTED IF
                    ;10388                                          ; ERROR
                    ;10389  LOOP.T11.7:
U 0B08, 2006.95     ;10390          MOV WR[3] TO WR[1]               ;EXPECTED DATA
U 0B09, 9812.F5     ;10391          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                    ;10392                                          ;SET UP TO WRITE TB
U 0B0A, B2F6.15     ;10393          WRITE.MEM LS[SHIFT.OS(4-0)]      ;WRITE TB WITH SHIFTING 1'S DATA
U 0B0B, 9C13.F5     ;10394          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                    ;10395                                          ;SET UP TO READ TB
U 0B0C, 3022.15     ;10396          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 0B0D, 0869.3C     ;10397          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 0B0E, 88B0.84     ;10398          JMP [LOOP.T11.7]                 ;ERROR LOOP IF ENABLED
                    ;10399
U 0B0F, A3C1.95     ;10400          ROL WR[3]                        ;NEXT EXPECTED DATA PATTERN
U 0B10, 7FF8.15     ;10401          INC LS[OS]                       ;INCREMENT INDEX FOR NEXT PATTERN
U 0B11, B6F8.15     ;10402          MOV LS[OS] TO WR[0]              ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                    ;10403          BIT LS[BIT5] WITH WR[0],         ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0B12, 594A.35     ;10404           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 0B13, 08B0.89     ;10405          JMP.IF[BITS.CLR] TO [LOOP.T11.7] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                    ;10406
U 0B14, E5F8.15     ;10407          CLR LS[OS]                       ;CLEAR INDEX
U 0B15, 0A18.2C     ;10408          JSR [NEXT.TB.ADDRESS]            ;CALCULATE NEXT TB ADDRESS
U 0B16, 88B0.84     ;10409          JMP [LOOP.T11.7]                 ;RETURN HERE IF NOT DONE ALL ADDRESSES
                    ;10410
U 0B17, FF82.15     ;10411          INC LS[ERROR.NUMBER]             ;ERROR 8
U 0B18, E5F8.15     ;10412          CLR LS[OS]                       ;CLEAR INDEX
U 0B19, 6512.15     ;10413          CLR LS[T9]                       ;CLEAR TB ADDRESS (ADDRESS STORED IN LS 9)
U 0B1A, 6588.15     ;10414          CLR LS[ADDRESS.DATA]             ;CLEAR ADDRESS INFO TO BE PRINTED IF
                    ;10415                                          ; ERROR
U 0B1B, 5F51.95     ;10416          MCOM LS[BIT8] TO WR[3]           ;FIRST EXPECTED DATA (RECEIVED DATA)
                    ;10417                                          ; IS SHIFTED LEFT 8 PLACES)
                    ;10418  REPEAT.T11.8:
U 0B1C, 5FF6.15     ;10419          MCOM LS[SHIFT.OS(4-0)] TO WR[0] ;GET CURRENT DATA PATTERN
U 0B1D, 3E10.15     ;10420          MOV WR[0] TO LS[T8]              ;PUT DATA PATTERN IN TEMPORARY LS
                    ;10421
```

```
                    ;10422  LOOP.T11.8:
U 0B1E, 2006,95     ;10423          MOV WR[3] TO WR[1]                  ;EXPECTED DATA
U 0B1F, 9812,F5     ;10424          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                    ;10425                                              ;SET UP TO WRITE TB
U 0B20, 3210,15     ;10426          WRITE.MEM LS[T8]                    ;WRITE TB WITH SHIFTING 0'S DATA
U 0B21, 9C13,F5     ;10427          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                    ;10428                                              ;SET UP TO READ TB
U 0B22, 3022,15     ;10429          MOV MEM.DATA TO WR[0]                ;GET RESULT
U 0B23, 0869,3C     ;10430          JSR [CHECK.RESULT]                  ;CHECK FOR ERRORS
U 0B24, 08B1,E4     ;10431          JMP [LOOP.T11.8]                    ;ERROR LOOP IF ENABLED
                    ;10432
U 0B25, A3C1,95     ;10433          ROL WR[3]                           ;NEXT EXPECTED DATA PATTERN
U 0B26, 7FF8,15     ;10434          INC LS[OS]                          ;INCREMENT INDEX FOR NEXT PATTERN
U 0B27, B6F8,15     ;10435          MOV LS[OS] TO WR[0]                 ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                    ;10436          BIT LS[BIT5] WITH WR[0],            ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0B28, 594A,35     ;10437           DT(LONG)&SET.ALU.CC                ; AND SET CONDITION CODES
U 0B29, 08B1,C9     ;10438          JMP.IF[BITS.CLR] TO [REPEAT.T11.8] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                    ;10439
U 0B2A, E5F8,15     ;10440          CLR LS[OS]                          ;CLEAR INDEX
U 0B2B, 0A18,2C     ;10441          JSR [NEXT.TB.ADDRESS]               ;CALCULATE NEXT TB ADDRESS
U 0B2C, 88B1,C4     ;10442          JMP [REPEAT.T11.8]                  ;RETURN HERE IF NOT DONE ALL ADDRESSES
                    ;10443
J 0B2D, FF82,15     ;10444          INC LS[ERROR.NUMBER]                ;ERROR 9
U 0B2E, 6512,15     ;10445          CLR LS[T9]                          ;CLEAR LS LOCATION CONTAINING ADDRESS
                    ;10446  BACK.T11.9:
U 0B2F, 9812,F5     ;10447          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                    ;10448                                              ;SET UP TO WRITE BACKGROUND IN TB
U 0B30, B29C,15     ;10449          WRITE.MEM LS[ZERO]                  ;PUT ZEROS IN TB
                    ;10450
U 0B31, 0A18,2C     ;10451          JSR [NEXT.TB.ADDRESS]               ;CALCULATE NEXT TB ADDRESS
U 0B32, 88B2,F4     ;10452          JMP [BACK.T11.9]                    ;RETURNS HERE IF NOT DONE YET
                    ;10453
U 0B33, 6588,15     ;10454          CLR LS[ADDRESS.DATA]                ;CLEAR ADDRESS INFO THAT GETS PRINTED
U 0B34, 6512,15     ;10455          CLR LS[T9]                          ;CLEAR TB ADDRESS STORAGE
U 0B35, 3651,95     ;10456          MOV LS[BIT8] TO WR[3]               ;SET BIT 8 OF WR 3.  ROL ON WR 3 WILL
                    ;10457                                              ; PRODUCE FIRST SHIFTED ADDRESS (1(H)).
U 0B36, B67F,15     ;10458          MOV LS[BIT31] TO WR[2]              ;SET BIT 31 OF WR 2.  ROL ON WR 2 WILL
                    ;10459                                              ; PRODUCE FIRST SHIFTED ADDRESS (1(H))
                    ;10460                                              ; FOR PRINTING IF ERROR
                    ;10461  LOOP.T11.9A:
U 0B37, 2F82,95     ;10462          CLR WR[1]                           ;EXPECTED DATA
U 0B38, 9C13,F5     ;10463          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                    ;10464                                              ;SET UP TO READ TB
U 0B39, 3022,15     ;10465          MOV MEM.DATA TO WR[0]                ;GET RESULT
U 0B3A, 0869,3C     ;10466          JSR [CHECK.RESULT]                  ;CHECK FOR ERRORS
U 0B3B, 88B3,74     ;10467          JMP [LOOP.T11.9A]                   ;ERROR LOOP IF ENABLED
                    ;10468
                    ;10469  LOOP.T11.9B:
U 0B3C, 369E,95     ;10470          MOV LS[ONES] TO WR[1]               ;EXPECTED DATA
U 0B3D, 9812,F5     ;10471          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                    ;10472                                              ;SET UP TO WRITE TB
U 0B3E, 329E,15     ;10473          WRITE.MEM LS[ONES]                  ;WRITE 1'S IN TB
U 0B3F, 9C13,F5     ;10474          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                    ;10475                                              ;SET UP TO READ TB FOR 1'S
U 0B40, 3022,15     ;10476          MOV MEM.DATA TO WR[0]                ;READ THE RESULT
```

```
U 0B41, 0869,3C ;10477          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0B42, 08B3,C4 ;10478          JMP [LOOP.T11.9B]               ;ERROR LOOP IF ENABLED
                ;10479
U 0B43, A3C1,95 ;10480          ROL WR[3]                       ;SHIFT ADDRESS TO PUT A 1 IN NEXT BIT
                ;10481          BIT LS[BIT0] WITH WR[3],        ;SEE IF BIT 0 IS SET (MEANS BIT 31 WAS
                ;10482                                          ; WAS JUST TESTED)
U 0B44, 5941,B5 ;10483           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0B45, 08B4,D1 ;10484          JMP.IF[BIT.SET] TO [TEST.T11.A] ;GO TO NEXT TEST IF DONE
                ;10485          BIT LS[BIT15] WITH WR[3],       ;SEE IF DONE BITS 5-0 (IF LAST TEST WAS
                ;10486                                          ; WITH BIT 14 SET, AFTER ROL BIT 15
                ;10487                                          ; WILL SET)
U 0B46, 595F,B5 ;10488           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 0B47, 5B00,09 ;10489          SKIP.IF[BITS.CLR]               ;SKIP NEXT INSTRUCTION IF BIT 15 NOT SET
U 0B48, 367F,95 ;10490          MOV LS[BIT31] TO WR[3]          ;ELSE SET BIT 31 FOR NEXT ADDRESS BIT
                ;10491                                          ; AND CLEAR BIT 15
U 0B49, BE13,95 ;10492          MOV WR[3] TO LS[T9]             ;STORE ADDRESS
U 0B4A, 23C1,15 ;10493          ROL WR[2]                       ;ADDRESS INFO TO PRINT IF ERROR
U 0B4B, 3E89,15 ;10494          MOV WR[2] TO LS[ADDRESS.DATA]   ;STORE IN LS
U 0B4C, 88B3,74 ;10495          JMP [LOOP.T11.9A]               ;REPEAT UNTIL ALL ADDRESS BITS HAVE BEEN
                ;10496                                          ; TESTED
                ;10497  TEST.T11.A:
U 0B4D, E580,15 ;10498          CLR LS[CONTROL.STATUS]          ;DISABLE 'OTHER' PRINTOUT
U 0B4E, FF82,15 ;10499          INC LS[ERROR.NUMBER]            ;ERROR A
U 0B4F, 3682,15 ;10500          MOV LS[ERROR.NUMBER] TO WR[0]   ;SAVE ERROR NUMBER
U 0B50, BE14,15 ;10501          MOV WR[0] TO LS[T10]            ;IN TEMPORARY LS LOCATION
U 0B51, 3653,15 ;10502          MOV LS[BIT9] TO WR[2]           ;SET UP INCREMENTER (INCREMENTS ADDRESS
                ;10503                                          ; BIT 0 AT RAM CHIP)
U 0B52, B653,95 ;10504          MOV LS[BIT9] TO WR[3]           ;BIT WE ARE GOING TO TOGGLE AT MAX RATE
                ;10505                                          ; FIRST BIT TO TOGGLE IS RAM BIT 0
U 0B53, 6512,15 ;10506          CLR LS[T9]                      ;CLEAR LS LOCATION CONTAINING ADDRESS
                ;10507  BACK.T11.A:
U 0B54, 9812,F5 ;10508          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                ;10509                                          ;SET UP TO WRITE BACKGROUND IN TB
U 0B55, B29C,15 ;10510          WRITE.MEM LS[ZERO]              ;PUT ZEROS IN TB
                ;10511
U 0B56, 0A18,2C ;10512          JSR [NEXT.TB.ADDRESS]           ;CALCULATE NEXT TB ADDRESS
U 0B57, 88B5,44 ;10513          JMP [BACK.T11.A]                ;RETURNS HERE IF NOT DONE YET
                ;10514
                ;10515  REPEAT.T11.A.1:
U 0B58, 6512,15 ;10516          CLR LS[T9]                      ;START AT ADDRESS 0 IN TB
U 0B59, 3614,15 ;10517          MOV LS[T10] TO WR[0]            ;GET ERROR NUMBER
U 0B5A, BE82,15 ;10518          MOV WR[0] TO LS[ERROR.NUMBER]   ;PUT IN LS FOR PRINTOUT
                ;10519  LOOP.T11.A.1:
U 0B5B, 2F82,95 ;10520          CLR WR[1]                       ;EXPECTED DATA
U 0B5C, 9C13,F5 ;10521          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                ;10522                                          ;SET UP TO READ TB
U 0B5D, 3022,15 ;10523          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0B5E, 0869,3C ;10524          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0B5F, 88B5,B4 ;10525          JMP [LOOP.T11.A.1]              ;ERROR LOOP IF ENABLED
                ;10526
U 0B60, 9812,F5 ;10527          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                ;10528                                          ;SET UP TO WRITE COMPLEMENT IN TB
U 0B61, 329E,15 ;10529          WRITE.MEM LS[ONES]              ;WRITE ONES IN THIS LS LOCATION
                ;10530
                ;10531          ADD WR[3] TO LS[T9],            ;TOGGLE BIT IN ADDRESS
```

```
U 0B62, EC13,D5 ;10532          DT(SIZE)&SET.ALU.CC             ; AND SET CONDITION CODES
U 0B63, 08B5,B0 ;10533          JMP.IF[N.CLR] TO [LOOP.T11.A.1] ;IF BIT 15 DIDN'T SET, REPEAT AT NEXT
                ;10534                                          ; ADDRESS
U 0B64, EC13,15 ;10535          ADD WR[2] TO LS[T9]             ;ELSE INCREMENT ADDRESS BELOW TOGGLED
                ;10536                                          ; BIT
U 0B65, B65E,15 ;10537          MOV LS[BIT15] TO WR[0]          ;GOING TO CLEAR BIT 15 IN LS 9
U 0B66, 6F12,15 ;10538          XOR WR[0] TO LS[T9]             ;CLEAR BIT 15 IN ADDRESS
                ;10539          BIT WR[3] WITH LS[T9],          ;IF TOGGLED BIT IS NOW SET, WE'RE DONE
U 0B67, D913,B5 ;10540           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 0B68, 08B5,B9 ;10541          JMP.IF[BITS.CLR] TO [LOOP.T11.A.1] ;REPEAT TEST AT NEW ADDRESS IF NOT
                ;10542                                          ; DONE
                ;10543
U 0B69, 3612,15 ;10544          MOV LS[T9] TO WR[0]             ;GET ADDRESS FROM LS
U 0B6A, C528,15 ;10545          BIC LS[#FFFF] TO WR[0]          ;CLEAR BITS 9 THROUGH 14 (ALONG WITH
                ;10546                                          ; OTHERS THAT ARE ALREADY CLEAR)
                ;10547          XOR LS[BIT31] TO WR[0],         ;TOGGLE BIT 31
U 0B6B, 437E,35 ;10548           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0B6C, BE12,15 ;10549          MOV WR[0] TO LS[T9]             ;PUT NEW ADDRESS IN LS 9
U 0B6D, 88B5,B8 ;10550          JMP.IF[N.SET] TO [LOOP.T11.A.1] ;IF BIT 31 WAS CLEAR, REPEAT USING SAME
                ;10551                                          ; TOGGLED BIT BUT WITH BIT 31 SET
                ;10552
                ;10553                                          ; ELSE GOING TO START DESCENDING ADDRS
U 0B6E, 6512,15 ;10554          CLR LS[T9]                      ;START AT ADDRESS 0 IN TB (WILL BE
                ;10555                                          ; COMPLEMENTED TO START AT TOP ADDRESS)
U 0B6F, FF82,15 ;10556          INC LS[ERROR.NUMBER]            ;ERROR B
                ;10557  REPEAT.T11.B.1:
U 0B70, 7D12,15 ;10558          COM LS[T9]                      ;COMPLEMENT ADDRESS TO GET DESCENDING
                ;10559                                          ; ADDRESSES
                ;10560  LOOP.T11.B.1:
U 0B71, 369E,95 ;10561          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 0B72, 9C13,F5 ;10562          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                ;10563                                          ;SET UP TO READ TB
U 0B73, 3022,15 ;10564          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0B74, 0869,3C ;10565          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0B75, 08B7,14 ;10566          JMP [LOOP.T11.B.1]              ;ERROR LOOP IF ENABLED
                ;10567
U 0B76, 9812,F5 ;10568          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                ;10569                                          ;SET UP TO WRITE COMPLEMENT IN TB
U 0B77, B29C,15 ;10570          WRITE.MEM LS[ZERO]              ;WRITE ZEROS IN THIS LS LOCATION
                ;10571
U 0B78, 7D12,15 ;10572          COM LS[T9]                      ;RECOMPLEMENT ADDRESS TO GET ORIGINAL
                ;10573          ADD WR[3] TO LS[T9],            ;TOGGLE BIT IN ADDRESS
U 0B79, EC13,D5 ;10574           DT(SIZE)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0B7A, 08B7,00 ;10575          JMP.IF[N.CLR] TO [REPEAT.T11.B.1] ;IF BIT 15 DIDN'T SET, REPEAT AT NEXT
                ;10576                                          ; ADDRESS
U 0B7B, EC13,15 ;10577          ADD WR[2] TO LS[T9]             ;ELSE INCREMENT ADDRESS BELOW TOGGLED
                ;10578                                          ; BIT
U 0B7C, B65E,15 ;10579          MOV LS[BIT15] TO WR[0]          ;GOING TO CLEAR BIT 15 IN LS 9
U 0B7D, 6F12,15 ;10580          XOR WR[0] TO LS[T9]             ;CLEAR BIT 15 IN ADDRESS
                ;10581          BIT WR[3] WITH LS[T9],          ;IF TOGGLED BIT IS NOW SET, WE'RE DONE
U 0B7E, D913,B5 ;10582           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 0B7F, 08B7,09 ;10583          JMP.IF[BITS.CLR] TO [REPEAT.T11.B.1] ;REPEAT TEST AT NEW ADDRESS IF NOT
                ;10584                                          ; DONE
                ;10585
U 0B80, 3612,15 ;10586          MOV LS[T9] TO WR[0]             ;GET ADDRESS FROM LS
```

```
U 0B81, C528,15 ;10587          BIC LS[#FFFF] TO WR[0]          ;CLEAR BITS 9 THROUGH 14 (ALONG WITH
                ;10588                                          ; OTHERS THAT ARE ALREADY CLEAR)
                ;10589          XOR LS[BIT31] TO WR[0],         ;TOGGLE BIT 31
U 0B82, 437E,35 ;10590           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0B83, BE12,15 ;10591          MOV WR[0] TO LS[T9]             ;PUT NEW ADDRESS IN LS 9
U 0B84, 88B7,08 ;10592          JMP.IF[N.SET] TO [REPEAT.T11.B.1] ;IF BIT 31 WAS CLEAR, REPEAT USING SAME
                ;10593                                          ; TOGGLED BIT BUT WITH BIT 31 SET
                ;10594
U 0B85, A3C1,95 ;10595          ROL WR[3]                       ;NEXT BIT TO TOGGLE AT MAX RATE
                ;10596          BIT LS[BIT15] WITH WR[3],       ;SEE IF BIT 15 HAS SET
U 0B86, 595F,B5 ;10597           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0B87, 08B5,89 ;10598          JMP.IF[BITS.CLR] TO [REPEAT.T11.A.1] ;REPEAT TEST WITH NEW TOGGLED BIT
                ;10599                                          ; IF BIT 15 HAS NOT SET
                ;10600                                          ; ELSE GOING TO TOGGLE BIT 31 NEXT
                ;10601
U 0B88, 367F,95 ;10602          MOV LS[BIT31] TO WR[3]          ;BIT WE ARE GOING TO TOGGLE AT MAX RATE
                ;10603                                          ; BIT TO TOGGLE NOW IS RAM BIT 6
U 0B89, 6512,15 ;10604          CLR LS[T9]                      ;START AT ADDRESS 0 IN TB
U 0B8A, 3614,15 ;10605          MOV LS[T10] TO WR[0]            ;GET ERROR NUMBER
U 0B8B, BE82,15 ;10606          MOV WR[0] TO LS[ERROR.NUMBER]   ;STORE IN LS FOR ERROR PRINTOUT
                ;10607  LOOP.T11.A1.1:
U 0B8C, 2F82,95 ;10608          CLR WR[1]                       ;EXPECTED DATA
U 0B8D, 9C13,F5 ;10609          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                ;10610                                          ;SET UP TO READ TB
U 0B8E, 3022,15 ;10611          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0B8F, 0869,3C ;10612          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0B90, 88B8,C4 ;10613          JMP [LOOP.T11.A1.1]             ;ERROR LOOP IF ENABLED
                ;10614
U 0B91, 9812,F5 ;10615          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                ;10616                                          ;SET UP TO WRITE COMPLEMENT IN TB
U 0B92, 329E,15 ;10617          WRITE.MEM LS[ONES]              ;WRITE ONES IN THIS LS LOCATION
                ;10618
                ;10619          XOR WR[3] TO LS[T9],            ;TOGGLE BIT IN ADDRESS
U 0B93, EF13,B5 ;10620           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0B94, 8888,C8 ;10621          JMP.IF[N.SET] TO [LOOP.T11.A1.1] ;IF BIT 31 DIDN'T CLEAR, REPEAT AT NEXT
                ;10622                                          ; ADDRESS
                ;10623          ADD WR[2] TO LS[T9],            ;ELSE INCREMENT ADDRESS BELOW TOGGLED
                ;10624                                          ; BIT
U 0B95, 6C13,55 ;10625           DT(SIZE)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0B96, 08B8,C0 ;10626          JMP.IF[N.CLR] TO [LOOP.T11.A1.1] ;REPEAT TEST AT NEW ADDRESS IF NOT
                ;10627                                          ; DONE
U 0B97, 6512,15 ;10628          CLR LS[T9]                      ;START AT ADDRESS 0 IN TB (WILL BE
                ;10629                                          ; COMPLEMENTED TO FORM TOP ADDRESS)
U 0B98, FF82,15 ;10630          INC LS[ERROR.NUMBER]            ;ERROR B
                ;10631  REPEAT.T11.B1.1:
U 0B99, 7D12,15 ;10632          COM LS[T9]                      ;COMPLEMENT ADDRESS TO DESCEND
                ;10633  LOOP.T11.B1.1:
U 0B9A, 369E,95 ;10634          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 0B9B, 9C13,F5 ;10635          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                ;10636                                          ;SET UP TO READ TB
U 0B9C, 3022,15 ;10637          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0B9D, 0869,3C ;10638          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0B9E, 0889,A4 ;10639          JMP [LOOP.T11.B1.1]             ;ERROR LOOP IF ENABLED
                ;10640
U 0B9F, 9812,F5 ;10641          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
```

```
                    ;10642                                  ;SET UP TO WRITE COMPLEMENT IN TB
U 0BA0, B29C,15     ;10643          WRITE.MEM LS[ZERO]      ;WRITE ZEROS IN THIS LS LOCATION
                    ;10644
U 0BA1, 7D12,15     ;10645          COM LS[T9]              ;RECOMPLEMENT ADDRESS TO GET ORIGINAL
                    ;10646          XOR WR[3] TO LS[T9],    ;TOGGLE BIT IN ADDRESS
U 0BA2, EF13,B5     ;10647           DT(LONG)&SET.ALU.CC    ; AND SET CONDITION CODES
U 0BA3, 08B9,98     ;10648          JMP.IF[N.SET] TO [REPEAT.T11.B1.1] ;IF BIT 31 DIDN'T CLEAR, REPEAT AT NEXT
                    ;10649                                  ; ADDRESS
                    ;10650          ADD WR[2] TO LS[T9],    ;ELSE INCREMENT ADDRESS BELOW TOGGLED
                    ;10651                                  ; BIT
U 0BA4, 6C13,55     ;10652           DT(SIZE)&SET.ALU.CC    ; AND SET CONDITION CODES
U 0BA5, 88B9,90     ;10653          JMP.IF[N.CLR] TO [REPEAT.T11.B1.1] ;REPEAT TEST AT NEW ADDRESS IF NOT DONE
                    ;10654                                  ; ELSE, DONE!!
                    ;10655  END.T11:
```

```
;10656  .PAGE   "  TEST 12 UNIBUS Map RAM Test (M8391  MCT Module)"
;10657  ;++
;10658  ;
;10659  ; Test Description:
;10660  ;
;10661  ;       This test checks the UNIBUS map portion of the MCT board 1KX4 TB RAMS.
;10662  ;       Addresses 200(H) to 3FF(H) can be accessed (512 decimal addresses)
;10663  ;       using the WRITE.UBS.MAP memory controller micro-routine.  The same
;10664  ;       addresses can be read back for verification using the READ.UBS.MAP
;10665  ;       routine.  Other addresses in the 1K by 4 RAM are either unused or
;10666  ;       tested in the preceeding test.  The address accessed in the TB RAM is
;10667  ;       determined by VAR bits 09 through 17.  RAM address bit 9 is forced to 1
;10668  ;       by the hardware.  VAR bits 0 through 8 and 18 through 31 have no effect
;10669  ;       on the UBS map RAM address selected.  The address is sent from LS
;10670  ;       during a MEM.REQ CPU instruction with the MF field = WRITE.UBS.MAP (0F)
;10671  ;       or READ.UBS.MAP (12H).  A CPU WRITE.MEM LS instruction sends the data
;10672  ;       to be written in the UBS map in the following manner:  LS bits 00-15
;10673  ;       go to PA bits 09-24 respectively, and LS bits 25-31 go to the byte
;10674  ;       offset, modify, prot A, prot B, prot C, prot D, and valid bits respec-
;10675  ;       tively.  The data to write in the UBS map is rotated in the data
;10676  ;       latches and stored in the ECC latches.  It is then read out of the
;10677  ;       latches and written into the TB RAM.  Note that PA 24 is not used, only
;10678  ;       PA 9-23 is written into the TB.
;10679  ;         The data path for the write is as follows:  A MEM.REQ is issued with
;10680  ;       MF=WRITE.UBS.MAP and DT=longword.  The dispatch  occurs as described in
;10681  ;       the beginning of this listing.  At the dispatch address, the trans-
;10682  ;       ceivers are set up to receive data from the CPU to write into the TB,
;10683  ;       and the transcievers on the MCT are set up to pass data from the MC bus
;10684  ;       to the PA bits and the other TB data bits. Memory busy is set, rotate
;10685  ;       byte is set, and TB MUX UNIBUS is set to enable UNIBUS address decod-
;10686  ;       ing.  The rest of the data path is identical to the WRITE.TB flow in
;10687  ;       the previous test, starting at the dispatch address.
;10688  ;         The read flows are initiated by a MEM.REQ with MF=READ.UBS.MAP (12H).
;10689  ;       The dispatch occurs as described in the beginning of this listing.  At
;10690  ;       the dispatch address, VAR bypass is enabled, memory busy is set, and
;10691  ;       the MCT transceiver that passes the TB data other than the PA bits is
;10692  ;       enabled.  TB MUX UNIBUS is also set to set up for UNIBUS addresses.
;10693  ;       The rest of the data path is identical to the READ.TB flow in the pre-
;10694  ;       ceeding test starting at the dispatch address.
;10695  ;         The above cycles will be used with data patterns of all 1's, all 0's,
;10696  ;       shifting 1,s and shifting 0's.  Then a shifting address bit test will
;10697  ;       be run to check address lines.  Finally a moving inversion test will be
;10698  ;       executed to check for slow address logic.  This test will run like a
;10699  ;       march test, except it is run 9 times, each time toggeling a different
;10700  ;       address bit at the maximum rate.  The march will write a background of
;10701  ;       zeros, then increment through the UBS map doing a read 0's, write 1's
;10702  ;       sequence.  When the last address has been checked, the march will de-
;10703  ;       crement through the UBS map executing a read 1's, write 0's sequence.
;10704  ;
;10705  ;
;10706  ; Assumptions:
;10707  ;
;10708  ;       It is assumed that all previous tests have run successfully.
;10709  ;
;10710  ; Test Steps:
```

```
;10711  ;
;10712  ;       1)  Set up error mask, error number, and module number in LS (for
;10713  ;           error printouts) and clear previous error number in LS.  The error
;10714  ;           mask will allow bits 1 through 22 to be checked.
;10715  ;       2)  Load WR 1 with all 1's, and execute a MEM.REQ with MF=WRITE.UBS.MAP
;10716  ;           and DT=longword.  Use an LS location containing 0's as the address
;10717  ;           (since RAM address bit 9 is forced high, the write will occur to
;10718  ;           address 200(H)).
;10719  ;       3)  Execute a WRITE.MEM LS with an LS location containing all 1's.
;10720  ;       4)  Execute a MEM.REQ with MF=READ.UBS.MAP and DT=longword.  Use an LS
;10721  ;           location containing 0's as the address (reads address 200(H)).
;10722  ;       5)  Execute a MOV MEM.DATA to WR[0] and check result for 1's.
;10723  ;       6)  Repeat steps 2 through 5 with data of 0's in step 3 and 5.
;10724  ;       7)  Repeat steps 2 through 5 with data of shifting 1's in step 3 and 5.
;10725  ;           Expected data in WR 1 is data sent shifted 8 bits left.
;10726  ;       8)  Repeat steps 2 through 5 with data of shifting 0's in step 3 and 5.
;10727  ;           Expected data in WR 1 is data sent shifted 8 bits left.
;10728  ;       9)  Repeat steps 2 through 5 with data patterns of 0's, 1's, shifting
;10729  ;           1's and shifting 0's, incrementing through all addresses with each
;10730  ;           data pattern.  This verifies all RAM cells that can be accessed.
;10731  ;           Expected data in WR 1 is data sent shifted 8 bits left.
;10732  ;       10) Execute a march type pattern while changing the address by shifting
;10733  ;           a 1 bit across an address field of zeros.  Write a background of
;10734  ;           0's, then read 0's, write 1's, read 0's.  This will be repeated 9
;10735  ;           times while shifting a 1 across the address lines.  This will ver-
;10736  ;           ify the address lines and gross failures in the RAM chip.
;10737  ;       11) Execute a moving inversion test (described in description above).
;10738  ;
;10739  ;
;10740  ; Errors:
;10741  ;
;10742  ;       NOTE:  OTHER data is UBS MAP address being tested.
;10743  ;
;10744  ;       Error 1 - Write or read UBS map failed with data of all ones, RAM
;10745  ;                 address 200(H).
;10746  ;       Error 2 - Write or read UBS map failed with data of all zeros, RAM
;10747  ;                 address 200(H).
;10748  ;       Error 3 - Write or read UBS map failed with data of shifting 1's, RAM
;10749  ;                 address 200(H).
;10750  ;       Error 4 - Write or read UBS map failed with data of shifting 0's, RAM
;10751  ;                 address 200(H).
;10752  ;       Error 5 - UBS map failure with data of all ones.
;10753  ;       Error 6 - UBS map failure with data of all zeros
;10754  ;       Error 7 - UBS map failure with data of shifting ones.
;10755  ;       Error 8 - UBS map failure with data of shifting zeros.
;10756  ;       Error 9 - UBS map address failure, shifting address bit test.
;10757  ;       Error A - UBS map failure with moving inversion pattern,ascending
;10758  ;                 addresses.
;10759  ;       Error B - UBS map failure with moving inversion pattern,descending
;10760  ;                 addresses.
;10761  ;
;10762  ; Debug:
;10763  ;
;10764  ;       ***NOTE***The UBS map address that failed will be printed under "other
;10765  ;       data" in the error report (except for errors A and B). The bits will be
```

```
                    ;10766  ;       arranged so as to give the actual RAM address in hexadecimal format.
                    ;10767  ;
                    ;10768  ;       Two methods of debug will be explained.  If the MCT module is in a
                    ;10769  ;       test station then the MCT can be single stepped making debug easier
                    ;10770  ;       in some cases.  Otherwise only the CPU can be single stepped and this
                    ;10771  ;       method will be explained also.  When single stepping the memory con-
                    ;10772  ;       troller will help in debug, this section will explicitly suggest it.
                    ;10773  ;
                    ;10774  ;       Error 1 through 8 - The most likely failure is the RAM chip itself,
                    ;10775  ;       since all the data inputs and control logic has been tested in the TB
                    ;10776  ;       RAM test preceeding.  Only the address is different, and would have no
                    ;10777  ;       effect on the data test.
                    ;10778  ;
                    ;10779  ;       Error 9 - This error indicates a problem with either the address lines
                    ;10780  ;       or an internal address problem within a chip.  If all bits are failing,
                    ;10781  ;       suspect an address line.  The address lines can be examined by enabling
                    ;10782  ;       SOMM mode at the MEM.REQ with MF=WRITE.UBS.MAP.  Each time the CPU
                    ;10783  ;       halts on the micro-match, the address lines can be examined on the MCT
                    ;10784  ;       module. A one should be shifting across the address lines from LVA 09
                    ;10785  ;       H through LVA 14 H and then to TBA 6 H, 7 H, and 8 H after a CONTINUE
                    ;10786  ;       is issued.  TBA 9 should always be high.  If TBA 6 H, 7 H, or 8 H fails
                    ;10787  ;       to go high, check the 4X2 MUX that supplies the TBA address as outputs.
                    ;10788  ;       The select signal UB TB SEL H should be high.  This can be traced back
                    ;10789  ;       to the negated input 'OR' gate with CPH/UBL H and UB TB SEL L as the
                    ;10790  ;       inputs.  UB TB SEL L should be low.  If not, suspect the PAL that gen-
                    ;10791  ;       erates it.
                    ;10792  ;         If all bits are not failing, suspect a bad cell in the RAM chip it-
                    ;10793  ;       self.  If a group of four bits is failing, suspect a break in the ad-
                    ;10794  ;       dress line etch to that RAM chip (most likely address line 7, 8, or 9).
                    ;10795  ;         One other possibility is LVA 15 H coming from the VAR bypass register
                    ;10796  ;       on the READ.UBS.MAP routine.  This bit was not used in the TB read
                    ;10797  ;       flow.  Loop on error can be used to check that this bit toggles cor-
                    ;10798  ;       rectly.
                    ;10799  ;
                    ;10800  ;       Error A and B - This error most likely indicates a bad RAM chip,
                    ;10801  ;       either in a single cell or in the address switching logic.
                    ;10802  ;         The address can be determined by examining LS location 9(H).  Bits
                    ;10803  ;       9-14 correspond to bits 0-5 at the RAM chip.  Bit 31 corresponds to
                    ;10804  ;       bit 6 at the RAM chip.
                    ;10805  ;
                    ;10806  ;--
                    ;10807
                    ;10808  T.12:
                    ;10809
U 0BA6, B65E,15     ;10810          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;10811                                          ; STATUS WORD
U 0BA7, 3E80,15     ;10812          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;10813                                          ; BIT 15 INDICATES START OF TEST TO
                    ;10814                                          ; CONSOLE
U 0BA8, 10E0,15     ;10815          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;10816  WAIT.T12.0:
U 0BA9, 08BA,94     ;10817          JMP [WAIT.T12.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;10818                                          ; RESPOND
                    ;10819
U 0BAA, 0A1A,AC     ;10820          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
```

```
                    ;10821
U 0BAB, B640,15     ;10822          MOV LS[#1] TO WR[0]              ;1 IN WR
U 0BAC, BEFC,15     ;10823          MOV WR[0] TO LS[SIZE]            ;SET UP SIZE REG FOR DATA TYPE OF WORD
U 0BAD, B62A,15     ;10824          MOV LS[#FF000000] TO WR[0]       ;SET TOP BYTE FOR ERROR MASK
U 0BAE, C76E,15     ;10825          BIS LS[BIT23] TO WR[0]           ;SET BIT 23
U 0BAF, C740,15     ;10826          BIS LS[BIT0] TO WR[0]            ;AND SET BIT 0.  NOW BITS 1 THROUGH 22
                    ;10827                                           ; WILL BE CHECKED FOR ERRORS
U 0BB0, 3E8A,15     ;10828          MOV WR[0] TO LS[ERROR.MASK]      ;STORE IN LS ERROR MASK
U 0BB1, B652,15     ;10829          MOV LS[#200] TO WR[0]            ;PUT A VALUE OF 200 IN WR 0
U 0BB2, BE88,15     ;10830          MOV WR[0] TO LS[ADDRESS.DATA]    ;LOAD A 200 TO BE PRINTED UNDER 'OTHER'
                    ;10831                                           ; IN THE ERROR REPORT, AS THE ADDRESS
U 0BB3, B670,15     ;10832          MOV LS[OTHER.DATA] TO WR[0]      ;SET BIT 24 IN WR 0
U 0BB4, 3E80,15     ;10833          MOV WR[0] TO LS[CONTROL.STATUS]  ;CONSOLE WILL PRINT UNDER 'OTHER' IF
                    ;10834                                           ; ERROR
                    ;10835
                    ;10836  LOOP.T12.1:
U 0BB5, 369E,95     ;10837          MOV LS[ONES] TO WR[1]            ;EXPECTED DATA
U 0BB6, 1B9D,F5     ;10838          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                    ;10839                                           ;SET UP TO WRITE UB MAP ADDRESS 200
U 0BB7, 329E,15     ;10840          WRITE.MEM LS[ONES]               ;WRITE ALL ONES IN UB MAP
U 0BB8, 1C9D,75     ;10841          MEM.REQ[READ.UBS.MAP] ADRS[#0] DT[LONG]
                    ;10842                                           ;SET UP TO READ UB MAP
U 0BB9, 3022,15     ;10843          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 0BBA, 0869,3C     ;10844          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 0BBB, 8888,54     ;10845          JMP [LOOP.T12.1]                 ;ERROR LOOP IF ENABLED
                    ;10846
U 0BBC, FF82,15     ;10847          INC LS[ERROR.NUMBER]             ;ERROR 2
                    ;10848  LOOP.T12.2:
U 0BBD, B69C,95     ;10849          MOV LS[ZERO] TO WR[1]            ;EXPECTED DATA
U 0BBE, 1B9D,F5     ;10850          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                    ;10851                                           ;SET UP TO WRITE UB MAP ADDRESS 200
U 0BBF, B29C,15     ;10852          WRITE.MEM LS[ZERO]               ;WRITE ALL ZEROS IN UB MAP
U 0BC0, 1C9D,75     ;10853          MEM.REQ[READ.UBS.MAP] ADRS[#0] DT[LONG]
                    ;10854                                           ;SET UP TO READ UB MAP
U 0BC1, 3022,15     ;10855          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 0BC2, 0869,3C     ;10856          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 0BC3, 08BB,D4     ;10857          JMP [LOOP.T12.2]                 ;ERROR LOOP IF ENABLED
                    ;10858
U 0BC4, FF82,15     ;10859          INC LS[ERROR.NUMBER]             ;ERROR 3
U 0BC5, E5F8,15     ;10860          CLR LS[OS]                       ;CLEAR INDEX
                    ;10861
U 0BC6, 3651,95     ;10862          MOV LS[BIT8] TO WR[3]            ;FIRST EXPECTED DATA (RECEIVED DATA)
                    ;10863                                           ; IS SHIFTED LEFT 8 PLACES)
                    ;10864  LOOP.T12.3:
U 0BC7, 2006,95     ;10865          MOV WR[3] TO WR[1]               ;EXPECTED DATA
U 0BC8, 1B9D,F5     ;10866          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                    ;10867                                           ;SET UP TO WRITE UB MAP ADDRESS 200
U 0BC9, B2F6,15     ;10868          WRITE.MEM LS[SHIFT.OS(4-0)]      ;WRITE UB MAP WITH SHIFTING 1'S DATA
U 0BCA, 1C9D,75     ;10869          MEM.REQ[READ.UBS.MAP] ADRS[#0] DT[LONG]
                    ;10870                                           ;SET UP TO READ UB MAP ADDRESS 200
U 0BCB, 3022,15     ;10871          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 0BCC, 0869,3C     ;10872          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 0BCD, 88BC,74     ;10873          JMP [LOOP.T12.3]                 ;ERROR LOOP IF ENABLED
                    ;10874
U 0BCE, A3C1,95     ;10875          ROL WR[3]                        ;NEXT EXPECTED DATA PATTERN
```

```
U 0BCF, 7FF8,15 ;10876          INC LS[OS]                          ;INCREMENT INDEX FOR NEXT PATTERN
U 0BD0, B6F8,15 ;10877          MOV LS[OS] TO WR[0]                 ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                ;10878          BIT LS[BIT5] WITH WR[0],            ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0BD1, 594A,35 ;10879           DT(LONG)&SET.ALU.CC                ; AND SET CONDITION CODES
U 0BD2, 08BC,79 ;10880          JMP.IF[BITS.CLR] TO [LOOP.T12.3] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                ;10881
U 0BD3, FF82,15 ;10882          INC LS[ERROR.NUMBER]                ;ERROR 4
U 0BD4, E5F8,15 ;10883          CLR LS[OS]                          ;CLEAR INDEX
U 0BD5, 5F51,95 ;10884          MCOM LS[BIT8] TO WR[3]              ;FIRST EXPECTED DATA (RECEIVED DATA)
                ;10885                                              ; IS SHIFTED LEFT 8 PLACES)
                ;10886  REPEAT.T12.4:
U 0BD6, 5FF6,15 ;10887          MCOM LS[SHIFT.OS(4-0)] TO WR[0] ;GET CURRENT DATA PATTERN
U 0BD7, 3E10,15 ;10888          MOV WR[0] TO LS[T8]                 ;PUT DATA PATTERN IN TEMPORARY LS
                ;10889  LOOP.T12.4:
U 0BD8, 2006,95 ;10890          MOV WR[3] TO WR[1]                  ;EXPECTED DATA
U 0BD9, 1B9D,F5 ;10891          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;10892                                              ;SET UP TO WRITE UB MAP ADDRESS 200
U 0BDA, 3210,15 ;10893          WRITE.MEM LS[T8]                    ;WRITE UB MAP WITH SHIFTING 0'S DATA
U 0BDB, 1C9D,75 ;10894          MEM.REQ[READ.UBS.MAP] ADRS[#0] DT[LONG]
                ;10895                                              ;SET UP TO READ UB MAP ADDRESS 200
U 0BDC, 3022,15 ;10896          MOV MEM.DATA TO WR[0]               ;GET RESULT
U 0BDD, 0869,3C ;10897          JSR [CHECK.RESULT]                  ;CHECK FOR ERRORS
U 0BDE, 08BD,84 ;10898          JMP [LOOP.T12.4]                    ;ERROR LOOP IF ENABLED
                ;10899
U 0BDF, A3C1,95 ;10900          ROL WR[3]                           ;NEXT EXPECTED DATA PATTERN
U 0BE0, 7FF8,15 ;10901          INC LS[OS]                          ;INCREMENT INDEX FOR NEXT PATTERN
U 0BE1, B6F8,15 ;10902          MOV LS[OS] TO WR[0]                 ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                ;10903          BIT LS[BIT5] WITH WR[0],            ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0BE2, 594A,35 ;10904           DT(LONG)&SET.ALU.CC                ; AND SET CONDITION CODES
U 0BE3, 08BD,69 ;10905          JMP.IF[BITS.CLR] TO [REPEAT.T12.4] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                ;10906
U 0BE4, FF82,15 ;10907          INC LS[ERROR.NUMBER]                ;ERROR 5
U 0BE5, 6512,15 ;10908          CLR LS[T9]                          ;CLEAR UB MAP ADDRESS (ADDRESS STORED
                ;10909                                              ; IN LS 9)
                ;10910  LOOP.T12.5:
U 0BE6, 369E,95 ;10911          MOV LS[ONES] TO WR[1]               ;EXPECTED DATA
U 0BE7, 1B13,F5 ;10912          MEM.REQ[WRITE.UBS.MAP] ADRS[T9] DT[LONG]
                ;10913                                              ;SET UP TO WRITE UB MAP
U 0BE8, 329E,15 ;10914          WRITE.MEM LS[ONES]                  ;WRITE ALL ONES IN UB MAP
U 0BE9, 1C13,75 ;10915          MEM.REQ[READ.UBS.MAP] ADRS[T9] DT[LONG]
                ;10916                                              ;SET UP TO READ UB MAP
U 0BEA, 3022,15 ;10917          MOV MEM.DATA TO WR[0]               ;GET RESULT
U 0BEB, 0869,3C ;10918          JSR [CHECK.RESULT]                  ;CHECK FOR ERRORS
U 0BEC, 88BE,64 ;10919          JMP [LOOP.T12.5]                    ;ERROR LOOP IF ENABLED
                ;10920
U 0BED, 8A18,CC ;10921          JSR [NEXT.UB.ADDRESS]               ;CALCULATE NEXT UB MAP ADDRESS
U 0BEE, 88BE,64 ;10922          JMP [LOOP.T12.5]                    ;RETURN HERE IF NOT DONE ALL ADDRESSES
                ;10923
U 0BEF, FF82,15 ;10924          INC LS[ERROR.NUMBER]                ;ERROR 6
U 0BF0, 6512,15 ;10925          CLR LS[T9]                          ;CLEAR UB MAP ADDRESS (ADDRESS STORED
                ;10926                                              ; IN LS 9)
U 0BF1, B652,15 ;10927          MOV LS[#200] TO WR[0]               ;PUT A VALUE OF 200 IN WR 0
U 0BF2, BE88,15 ;10928          MOV WR[0] TO LS[ADDRESS.DATA]       ;LOAD A 200 TO BE PRINTED UNDER 'OTHER'
                ;10929                                              ; IN THE ERROR REPORT, AS THE ADDRESS
                ;10930  LOOP.T12.6:
```

```
U 0BF3, B69C,95 ;10931          MOV LS[ZERO] TO WR[1]              ;EXPECTED DATA
U 0BF4, 1B13,F5 ;10932          MEM.REQ[WRITE.UBS.MAP] ADRS[T9] DT[LONG]
                ;10933                                            ;SET UP TO WRITE UB MAP
U 0BF5, B29C,15 ;10934          WRITE.MEM LS[ZERO]                ;WRITE ALL ZEROS IN UB MAP
U 0BF6, 1C13,75 ;10935          MEM.REQ[READ.UBS.MAP] ADRS[T9] DT[LONG]
                ;10936                                            ;SET UP TO READ UB MAP
U 0BF7, 3022,15 ;10937          MOV MEM.DATA TO WR[0]             ;GET RESULT
U 0BF8, 0869,3C ;10938          JSR [CHECK.RESULT]                ;CHECK FOR ERRORS
U 0BF9, 08BF,34 ;10939          JMP [LOOP.T12.6]                  ;ERROR LOOP IF ENABLED
                ;10940
U 0BFA, 8A18,CC ;10941          JSR [NEXT.UB.ADDRESS]             ;CALCULATE NEXT UB MAP ADDRESS
U 0BFB, 08BF,34 ;10942          JMP [LOOP.T12.6]                  ;RETURN HERE IF NOT DONE ALL ADDRESSES
                ;10943
U 0BFC, FF82,15 ;10944          INC LS[ERROR.NUMBER]              ;ERROR 7
U 0BFD, E5F8,15 ;10945          CLR LS[OS]                        ;CLEAR INDEX
U 0BFE, 6512,15 ;10946          CLR LS[T9]                        ;CLEAR UB MAP ADDRESS (ADDRESS STORED IN LS 9)
                ;10947
U 0BFF, 3651,95 ;10948          MOV LS[BIT8] TO WR[3]             ;FIRST EXPECTED DATA (RECEIVED DATA)
U 0C00, B652,15 ;10949          MOV LS[#200] TO WR[0]             ;PUT A VALUE OF 200 IN WR 0
U 0C01, BE88,15 ;10950          MOV WR[0] TO LS[ADDRESS.DATA]     ;LOAD A 200 TO BE PRINTED UNDER 'OTHER'
                ;10951                                            ; IN THE ERROR REPORT, AS THE ADDRESS
                ;10952  LOOP.T12.7:
U 0C02, 2006,95 ;10953          MOV WR[3] TO WR[1]                ;EXPECTED DATA
U 0C03, 1B13,F5 ;10954          MEM.REQ[WRITE.UBS.MAP] ADRS[T9] DT[LONG]
                ;10955                                            ;SET UP TO WRITE UB MAP
U 0C04, B2F6,15 ;10956          WRITE.MEM LS[SHIFT.OS(4-0)]       ;WRITE UB MAP WITH SHIFTING 1'S DATA
U 0C05, 1C13,75 ;10957          MEM.REQ[READ.UBS.MAP] ADRS[T9] DT[LONG]
                ;10958                                            ;SET UP TO READ UB MAP
U 0C06, 3022,15 ;10959          MOV MEM.DATA TO WR[0]             ;GET RESULT
U 0C07, 0869,3C ;10960          JSR [CHECK.RESULT]                ;CHECK FOR ERRORS
U 0C08, 08C0,24 ;10961          JMP [LOOP.T12.7]                  ;ERROR LOOP IF ENABLED
                ;10962
U 0C09, A3C1,95 ;10963          ROL WR[3]                         ;NEXT EXPECTED DATA PATTERN
U 0C0A, 7FF8,15 ;10964          INC LS[OS]                        ;INCREMENT INDEX FOR NEXT PATTERN
U 0C0B, B6F8,15 ;10965          MOV LS[OS] TO WR[0]               ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                ;10966          BIT LS[BIT5] WITH WR[0],          ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0C0C, 594A,35 ;10967           DT(LONG)&SET.ALU.CC              ; AND SET CONDITION CODES
U 0C0D, 88C0,29 ;10968          JMP.IF[BITS.CLR] TO [LOOP.T12.7] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                ;10969
U 0C0E, E5F8,15 ;10970          CLR LS[OS]                        ;CLEAR INDEX
U 0C0F, 8A18,CC ;10971          JSR [NEXT.UB.ADDRESS]             ;CALCULATE NEXT UB MAP ADDRESS
U 0C10, 08C0,24 ;10972          JMP [LOOP.T12.7]                  ;RETURN HERE IF NOT DONE ALL ADDRESSES
                ;10973
U 0C11, FF82,15 ;10974          INC LS[ERROR.NUMBER]              ;ERROR 8
U 0C12, E5F8,15 ;10975          CLR LS[OS]                        ;CLEAR INDEX
U 0C13, 6512,15 ;10976          CLR LS[T9]                        ;CLEAR UB MAP ADDRESS (ADDRESS STORED IN LS 9)
U 0C14, B652,15 ;10977          MOV LS[#200] TO WR[0]             ;PUT A VALUE OF 200 IN WR 0
U 0C15, BE88,15 ;10978          MOV WR[0] TO LS[ADDRESS.DATA]     ;LOAD A 200 TO BE PRINTED UNDER 'OTHER'
                ;10979                                            ; IN THE ERROR REPORT, AS THE ADDRESS
U 0C16, 5F51,95 ;10980          MCOM LS[BIT8] TO WR[3]            ;FIRST EXPECTED DATA (RECEIVED DATA)
                ;10981                                            ; IS SHIFTED LEFT 8 PLACES)
                ;10982  REPEAT.T12.8:
U 0C17, 5FF6,15 ;10983          MCOM LS[SHIFT.OS(4-0)] TO WR[0] ;GET CURRENT DATA PATTERN
U 0C18, 3E10,15 ;10984          MOV WR[0] TO LS[T8]               ;PUT DATA PATTERN IN TEMPORARY LS
                ;10985
```

```
                    ;10986  LOOP.T12.8:
U 0C19, 2006,95     ;10987          MOV WR[3] TO WR[1]                  ;EXPECTED DATA
U 0C1A, 1B13,F5     ;10988          MEM.REQ[WRITE.UBS.MAP] ADRS[T9] DT[LONG]
                    ;10989                                              ;SET UP TO WRITE UB MAP
U 0C1B, 3210,15     ;10990          WRITE.MEM LS[T8]                    ;WRITE UB MAP WITH SHIFTING 0'S DATA
U 0C1C, 1C13,75     ;10991          MEM.REQ[READ.UBS.MAP] ADRS[T9] DT[LONG]
                    ;10992                                              ;SET UP TO READ UB MAP
U 0C1D, 3022,15     ;10993          MOV MEM.DATA TO WR[0]               ;GET RESULT
U 0C1E, 0869,3C     ;10994          JSR [CHECK.RESULT]                  ;CHECK FOR ERRORS
U 0C1F, 08C1,94     ;10995          JMP [LOOP.T12.8]                    ;ERROR LOOP IF ENABLED
                    ;10996
U 0C20, A3C1,95     ;10997          ROL WR[3]                           ;NEXT EXPECTED DATA PATTERN
U 0C21, 7FF8,15     ;10998          INC LS[OS]                          ;INCREMENT INDEX FOR NEXT PATTERN
U 0C22, B6F8,15     ;10999          MOV LS[OS] TO WR[0]                 ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                    ;11000          BIT LS[BIT5] WITH WR[0],            ;SEE IF OS HAS INCREMENTED TO 20(H)
U 0C23, 594A,35     ;11001           DT(LONG)&SET.ALU.CC                ; AND SET CONDITION CODES
U 0C24, 08C1,79     ;11002          JMP.IF[BITS.CLR] TO [REPEAT.T12.8] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                    ;11003
U 0C25, E5F8,15     ;11004          CLR LS[OS]                          ;CLEAR INDEX
U 0C26, 8A18,CC     ;11005          JSR [NEXT.UB.ADDRESS]               ;CALCULATE NEXT UB MAP ADDRESS
U 0C27, 88C1,74     ;11006          JMP [REPEAT.T12.8]                  ;RETURN HERE IF NOT DONE ALL ADDRESSES
                    ;11007
U 0C28, FF82,15     ;11008          INC LS[ERROR.NUMBER]                ;ERROR 9
U 0C29, 6512,15     ;11009          CLR LS[T9]                          ;CLEAR LS LOCATION CONTAINING ADDRESS
                    ;11010  BACK.T12.9:
U 0C2A, 1B13,F5     ;11011          MEM.REQ[WRITE.UBS.MAP] ADRS[T9] DT[LONG]
                    ;11012                                              ;SET UP TO WRITE BACKGROUND IN UB MAP
U 0C2B, B29C,15     ;11013          WRITE.MEM LS[ZERO]                  ;PUT ZEROS IN UB MAP
                    ;11014
U 0C2C, 8A18,CC     ;11015          JSR [NEXT.UB.ADDRESS]               ;CALCULATE NEXT UB MAP ADDRESS
U 0C2D, 08C2,A4     ;11016          JMP [BACK.T12.9]                    ;RETURNS HERE IF NOT DONE YET
                    ;11017
U 0C2E, B652,15     ;11018          MOV LS[#200] TO WR[0]               ;PUT A VALUE OF 200 IN WR 0
U 0C2F, BE88,15     ;11019          MOV WR[0] TO LS[ADDRESS.DATA]       ;LOAD A 200 TO BE PRINTED UNDER 'OTHER'
                    ;11020                                              ; IN THE ERROR REPORT, AS THE ADDRESS
U 0C30, 6512,15     ;11021          CLR LS[T9]                          ;CLEAR UB MAP ADDRESS STORAGE
U 0C31, 3651,95     ;11022          MOV LS[BIT8] TO WR[3]               ;SET BIT 8 OF WR 3.  ROL ON WR 3 WILL
                    ;11023                                              ; PRODUCE FIRST SHIFTED ADDRESS (1(H)).
U 0C32, B67F,15     ;11024          MOV LS[BIT31] TO WR[2]              ;SET BIT 31 OF WR 2.  ROL ON WR 2 WILL
                    ;11025                                              ; PRODUCE FIRST SHIFTED ADDRESS (201(H))
                    ;11026                                              ; FOR PRINTING IF ERROR (AFTER A BIS ON
                    ;11027                                              ; BIT 9)
                    ;11028  LOOP.T12.9A:
U 0C33, 2F82,95     ;11029          CLR WR[1]                           ;EXPECTED DATA
U 0C34, 1C13,75     ;11030          MEM.REQ[READ.UBS.MAP] ADRS[T9] DT[LONG]
                    ;11031                                              ;SET UP TO READ UB MAP
U 0C35, 3022,15     ;11032          MOV MEM.DATA TO WR[0]               ;GET RESULT
U 0C36, 0869,3C     ;11033          JSR [CHECK.RESULT]                  ;CHECK FOR ERRORS
U 0C37, 88C3,34     ;11034          JMP [LOOP.T12.9A]                   ;ERROR LOOP IF ENABLED
                    ;11035
                    ;11036  LOOP.T12.9B:
U 0C38, 369E,95     ;11037          MOV LS[ONES] TO WR[1]               ;EXPECTED DATA
U 0C39, 1B13,F5     ;11038          MEM.REQ[WRITE.UBS.MAP] ADRS[T9] DT[LONG]
                    ;11039                                              ;SET UP TO WRITE UB MAP
U 0C3A, 329E,15     ;11040          WRITE.MEM LS[ONES]                  ;WRITE 1'S IN UB MAP
```

```
U 0C3B, 1C13,75 ;11041          MEM.REQ[READ.UBS.MAP] ADRS[T9] DT[LONG]
                ;11042                                          ;SET UP TO READ UB MAP FOR 1'S
U 0C3C, 3022,15 ;11043          MOV MEM.DATA TO WR[0]           ;READ THE RESULT
U 0C3D, 0869,3C ;11044          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0C3E, 08C3,84 ;11045          JMP [LOOP.T12.9B]               ;ERROR LOOP IF ENABLED
                ;11046
U 0C3F, A3C1,95 ;11047          ROL WR[3]                       ;SHIFT ADDRESS TO PUT A 1 IN NEXT BIT
U 0C40, BE13,95 ;11048          MOV WR[3] TO LS[T9]             ;STORE ADDRESS
U 0C41, 23C1,15 ;11049          ROL WR[2]                       ;ADDRESS INFO TO PRINT IF ERROR
U 0C42, 3E89,15 ;11050          MOV WR[2] TO LS[ADDRESS.DATA]   ;STORE IN LS
U 0C43, B652,15 ;11051          MOV LS[#200] TO WR[0]           ;GOING TO SET BIT 9 IN ADDRESS PRINTED
U 0C44, ED88,15 ;11052          BIS WR[0] TO LS[ADDRESS.DATA]   ;NOW TB RAM ADDRESS IS CORRECT
                ;11053          BIT LS[BIT18] WITH WR[3],       ;SEE IF DONE BITS 8-0 (IF LAST TEST WAS
                ;11054                                          ; WITH BIT 17 SET, AFTER ROL BIT 18
                ;11055                                          ; WILL SET)
U 0C45, 5965,B5 ;11056           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 0C46, 08C3,39 ;11057          JMP.IF[BITS.CLR] TO [LOOP.T12.9A] ;REPEAT TEST IF BIT 18 NOT SET WITH
                ;11058                                          ; NEXT SHIFTED ADDRESS
                ;11059
                ;11060  TEST.T12.A:
U 0C47, E580,15 ;11061          CLR LS[CONTROL.STATUS]          ;DISABLE 'OTHER' PRINTOUT
U 0C48, FF82,15 ;11062          INC LS[ERROR.NUMBER]            ;ERROR A
U 0C49, 3682,15 ;11063          MOV LS[ERROR.NUMBER] TO WR[0]   ;SAVE ERROR NUMBER
U 0C4A, BE14,15 ;11064          MOV WR[0] TO LS[T10]            ;IN TEMPORARY LS LOCATION
U 0C4B, 3653,15 ;11065          MOV LS[BIT9] TO WR[2]           ;SET UP INCREMENTER (INCREMENTS ADDRESS
                ;11066                                          ; BIT 0 AT RAM CHIP)
U 0C4C, B653,95 ;11067          MOV LS[BIT9] TO WR[3]           ;BIT WE ARE GOING TO TOGGLE AT MAX RATE
                ;11068                                          ; FIRST BIT TO TOGGLE IS RAM BIT 0
U 0C4D, 6512,15 ;11069          CLR LS[T9]                      ;CLEAR LS LOCATION CONTAINING ADDRESS
                ;11070  BACK.T12.A:
U 0C4E, 1B13,F5 ;11071          MEM.REQ[WRITE.UBS.MAP] ADRS[T9] DT[LONG]
                ;11072                                          ;SET UP TO WRITE BACKGROUND IN UB MAP
U 0C4F, B29C,15 ;11073          WRITE.MEM LS[ZERO]              ;PUT ZEROS IN UB MAP
                ;11074
U 0C50, 8A18,CC ;11075          JSR [NEXT.UB.ADDRESS]           ;CALCULATE NEXT UB MAP ADDRESS
U 0C51, 88C4,E4 ;11076          JMP [BACK.T12.A]                ;RETURNS HERE IF NOT DONE YET
                ;11077
                ;11078  REPEAT.T12.A.1:
U 0C52, 6512,15 ;11079          CLR LS[T9]                      ;START AT ADDRESS 200 IN UB MAP
U 0C53, 3614,15 ;11080          MOV LS[T10] TO WR[0]            ;GET ERROR NUMBER
U 0C54, BE82,15 ;11081          MOV WR[0] TO LS[ERROR.NUMBER]   ;PUT IN LS FOR PRINTOUT
                ;11082  LOOP.T12.A.1:
U 0C55, 2F82,95 ;11083          CLR WR[1]                       ;EXPECTED DATA
U 0C56, 1C13,75 ;11084          MEM.REQ[READ.UBS.MAP] ADRS[T9] DT[LONG]
                ;11085                                          ;SET UP TO READ UB MAP
U 0C57, 3022,15 ;11086          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0C58, 0869,3C ;11087          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0C59, 88C5,54 ;11088          JMP [LOOP.T12.A.1]              ;ERROR LOOP IF ENABLED
                ;11089
U 0C5A, 1B13,F5 ;11090          MEM.REQ[WRITE.UBS.MAP] ADRS[T9] DT[LONG]
                ;11091                                          ;SET UP TO WRITE COMPLEMENT IN UB MAP
U 0C5B, 329E,15 ;11092          WRITE.MEM LS[ONES]              ;WRITE ONES IN THIS LS LOCATION
                ;11093
U 0C5C, 6C13,95 ;11094          ADD WR[3] TO LS[T9]             ;TOGGLE BIT IN ADDRESS
U 0C5D, B664,15 ;11095          MOV LS[BIT18] TO WR[0]          ;GOING TO CHECK IF BIT 18 IS SET
```

```
                ;11096          BIT WR[0] WITH LS[T9],          ;CHECK BIT 18
U 0C5E, D912,35 ;11097           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0C5F, 08C5,59 ;11098          JMP.IF[BITS.CLR] TO [LOOP.T12.A.1] ;IF BIT 18 DIDN'T SET, REPEAT AT NEXT
                ;11099                                          ; ADDRESS
U 0C60, EC13,15 ;11100          ADD WR[2] TO LS[T9]             ;ELSE INCREMENT ADDRESS BELOW TOGGLED
                ;11101                                          ; BIT
U 0C61, B664,15 ;11102          MOV LS[BIT18] TO WR[0]          ;GOING TO CLEAR BIT 18 IN LS 9
U 0C62, 6F12,15 ;11103          XOR WR[0] TO LS[T9]             ;CLEAR BIT 18 IN ADDRESS
                ;11104          BIT WR[3] WITH LS[T9],          ;IF TOGGLED BIT IS NOW SET, WE'RE DONE
U 0C63, D913,B5 ;11105           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 0C64, 08C5,59 ;11106          JMP.IF[BITS.CLR] TO [LOOP.T12.A.1] ;REPEAT TEST AT NEW ADDRESS IF NOT
                ;11107                                          ; DONE
                ;11108                                          ; ELSE GOING TO START DESCENDING ADDRS
                ;11109
U 0C65, 6512,15 ;11110          CLR LS[T9]                      ;START AT ADDRESS 200 IN UB MAP (WILL BE
                ;11111                                          ; COMPLEMENTED TO START AT TOP ADDRESS)
U 0C66, FF82,15 ;11112          INC LS[ERROR.NUMBER]            ;ERROR B
                ;11113  REPEAT.T12.B.1:
U 0C67, 7D12,15 ;11114          COM LS[T9]                      ;COMPLEMENT ADDRESS TO GET DESCENDING
                ;11115                                          ; ADDRESSES
                ;11116  LOOP.T12.B.1:
U 0C68, 369E,95 ;11117          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 0C69, 1C13,75 ;11118          MEM.REQ[READ.UBS.MAP] ADRS[T9] DT[LONG]
                ;11119                                          ;SET UP TO READ UB MAP
U 0C6A, 3022,15 ;11120          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 0C6B, 0869,3C ;11121          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0C6C, 08C6,84 ;11122          JMP [LOOP.T12.B.1]              ;ERROR LOOP IF ENABLED
                ;11123
U 0C6D, 1B13,F5 ;11124          MEM.REQ[WRITE.UBS.MAP] ADRS[T9] DT[LONG]
                ;11125                                          ;SET UP TO WRITE COMPLEMENT IN UB MAP
U 0C6E, B29C,15 ;11126          WRITE.MEM LS[ZERO]              ;WRITE ZEROS IN THIS LS LOCATION
                ;11127
U 0C6F, 7D12,15 ;11128          COM LS[T9]                      ;RECOMPLEMENT ADDRESS TO GET ORIGINAL
U 0C70, 6C13,95 ;11129          ADD WR[3] TO LS[T9]             ;TOGGLE BIT IN ADDRESS
U 0C71, B664,15 ;11130          MOV LS[BIT18] TO WR[0]          ;GOING TO CHECK IF BIT 18 IS SET
                ;11131          BIT WR[0] WITH LS[T9],          ;SEE IF BIT 18 SET
U 0C72, D912,35 ;11132           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0C73, 88C6,79 ;11133          JMP.IF[BITS.CLR] TO [REPEAT.T12.B.1] ;IF BIT 18 DIDN'T SET, REPEAT AT NEXT
                ;11134                                          ; ADDRESS
U 0C74, EC13,15 ;11135          ADD WR[2] TO LS[T9]             ;ELSE INCREMENT ADDRESS BELOW TOGGLED
                ;11136                                          ; BIT
U 0C75, B664,15 ;11137          MOV LS[BIT18] TO WR[0]          ;GOING TO CLEAR BIT 18 IN LS 9
U 0C76, 6F12,15 ;11138          XOR WR[0] TO LS[T9]             ;CLEAR BIT 18 IN ADDRESS
                ;11139          BIT WR[3] WITH LS[T9],          ;IF TOGGLED BIT IS NOW SET, WE'RE DONE
U 0C77, D913,B5 ;11140           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 0C78, 88C6,79 ;11141          JMP.IF[BITS.CLR] TO [REPEAT.T12.B.1] ;REPEAT TEST AT NEW ADDRESS IF NOT
                ;11142                                          ; DONE
                ;11143                                          ; ELSE CHANGE TOGGLED BIT
U 0C79, A3C1,95 ;11144          ROL WR[3]                       ;NEXT BIT TO TOGGLE AT MAX RATE
                ;11145          BIT LS[BIT18] WITH WR[3],       ;SEE IF BIT 18 HAS SET
U 0C7A, 5965,B5 ;11146           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0C7B, 88C5,29 ;11147          JMP.IF[BITS.CLR] TO [REPEAT.T12.A.1] ;REPEAT TEST WITH NEW TOGGLED BIT
                ;11148                                          ; IF BIT 18 HAS NOT SET
                ;11149                                          ; ELSE, DONE'!
                ;11150  END.T12:
```

```
;11151  .PAGE   "  TEST 13 TB Step Test (M8391  MCT Module)"
;11152  ;++
;11153  ;
;11154  ; Test Description:
;11155  ;
;11156  ;       This test checks the TB step routine.  TB step uses the same logic as
;11157  ;       WRITE.TB, but takes a slightly different u-code flow.  This test is
;11158  ;       mainly to check the micro-store proms as no new logic is being tested.
;11159  ;         The data path for the TB step is almost identical to the WRITE.TB
;11160  ;       data path and flow.  The dispatch address differs from the WRITE.TB
;11161  ;       flow in that it sets DATIP instead of DATI.  This sets IC0 so that the
;11162  ;       branch at the end of the flow is taken.  At that branch, the VAR is
;11163  ;       incremented and TB write enable is turned off.  The rest of the word
;11164  ;       repeats the events in the cycle that branches on IC0.  The next cycle
;11165  ;       sets DATI so that IC0 is cleared and repeats the previous events except
;11166  ;       the VAR is not incremented.  The flow then returns to the normal WRITE
;11167  ;       .TB micro code at the cycle immediately preceeding the branch on IC0.
;11168  ;       Thus the TB is written again at a new address.
;11169  ;
;11170  ; Assumptions:
;11171  ;
;11172  ;       It is assumed that the previous tests have all run successfully.
;11173  ;
;11174  ; Test Steps:
;11175  ;
;11176  ;       1)  Set up error mask, error number, and module number in LS (for
;11177  ;           error printouts) and clear previous error number in LS.  The error
;11178  ;           mask will allow bits 1 through 22 to be checked.
;11179  ;       2)  Load an LS location with data containing 1's in the lower 9 bits,
;11180  ;           0's in all other bits.
;11181  ;       3)  Load WR 1 with all 1's, and execute a MEM.REQ with MF=WRITE.TB and
;11182  ;           DT=longword using the LS location loaded in step 2.
;11183  ;       4)  Execute a WRITE.MEM LS with an LS location containing all 1's.
;11184  ;       5)  Increment the LS location to write the next TB address and repeat
;11185  ;           steps 3 and 4.
;11186  ;       6)  Repeat step 2 and execute a MEM.REQ with MF=READ.TB and DT=longword
;11187  ;           using the LS location loaded in step 2.
;11188  ;       7)  Execute a MOV MEM.DATA to WR[0] and check result for 1's.  Repeat
;11189  ;           steps 6 and 7 to check the next location.
;11190  ;       8)  Load WR 1 with 0's, repeat step 2, and execute a MEM.REQ with MF=
;11191  ;           WRITE.TB.STEP, DT=longword.
;11192  ;       9)  Execute a WRITE.MEM LS with an LS location containing all 0's.
;11193  ;       10) Repeat steps 6 and 7 to assure that 2 TB locations were written.
;11194  ;
;11195  ; Errors:
;11196  ;
;11197  ;       Error 1 - Write or read TB failed (this error should not occur).
;11198  ;       Error 2 - Write TB step failed to write 2 consecutive TB locations
;11199  ;                 correctly.  OTHER data is TB address that failed.
;11200  ;
;11201  ; Debug:
;11202  ;
;11203  ;       Two methods of debug will be explained.  If the MCT module is in a
;11204  ;       test station then the MCT can be single stepped making debug easier
;11205  ;       in some cases.  Otherwise only the CPU can be single stepped and this
```

```
                    ;11206  ;       method will be explained also.  When single stepping the memory con-
                    ;11207  ;       troller will help in debug, this section will explicitly suggest it.
                    ;11208  ;
                    ;11209  ;       Error 1 - This error indicates a problem with a basic TB write and
                    ;11210  ;       read that was previously tested.  Run the TB RAM test again.
                    ;11211  ;
                    ;11212  ;       Error 2 - This error indicates a possible problem with the micro-code
                    ;11213  ;       in the control store PROM.  All logic used by this test has been pre-
                    ;11214  ;       viously tested.
                    ;11215  ;
                    ;11216  ;--
                    ;11217
                    ;11218  T.13:
U 0C7C, B65E,15     ;11219          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;11220                                          ; STATUS WORD
U 0C7D, 3E80,15     ;11221          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;11222                                          ; BIT 15 INDICATES START OF TEST TO
                    ;11223                                          ; CONSOLE
U 0C7E, 10E0,15     ;11224          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;11225  WAIT.T13.0:
U 0C7F, 08C7,F4     ;11226          JMP [WAIT.T13.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;11227                                          ; RESPOND
                    ;11228
U 0C80, 0A1A,AC     ;11229          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;11230
U 0C81, B62A,15     ;11231          MOV LS[#FF000000] TO WR[0]      ;SET TOP BYTE FOR ERROR MASK
U 0C82, C76E,15     ;11232          BIS LS[BIT23] TO WR[0]          ;ALSO SET BIT 23
U 0C83, C740,15     ;11233          BIS LS[BIT0] TO WR[0]           ;AND BIT 0
U 0C84, 3E8A,15     ;11234          MOV WR[0] TO LS[ERROR.MASK]     ;WILL CHECK BITS 1 THROUGH 22
                    ;11235  LOOP.T13.1:
U 0C85, 3627,15     ;11236          MOV LS[#FF] TO WR[2]            ;SET BITS 0-7 IN WR
U 0C86, C751,15     ;11237          BIS LS[BIT8] TO WR[2]           ;NOW BITS 0-8 ARE SET
U 0C87, 3E13,15     ;11238          MOV WR[2] TO LS[T9]             ;STORE THIS AS TB ADDRESS IN LS 9
                    ;11239
U 0C88, 9812,F5     ;11240          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                    ;11241                                          ;SET UP TO WRITE DATA AT TB ADDRESS 0
U 0C89, 329E,15     ;11242          WRITE.MEM LS[ONES]              ;PUT ONES IN TB ADDRESS 0
U 0C8A, FF12,15     ;11243          INC LS[T9]                      ;TB ADDRESS 1 IN LS 9 NOW
U 0C8B, 9812,F5     ;11244          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                    ;11245                                          ;SET UP TO WRITE DATA AT TB ADDRESS 1
U 0C8C, 329E,15     ;11246          WRITE.MEM LS[ONES]              ;PUT ONES IN TB ADDRESS 1
                    ;11247
                    ;11248
U 0C8D, 369E,95     ;11249          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 0C8E, 3E13,15     ;11250          MOV WR[2] TO LS[T9]             ;TB ADDRESS 0
U 0C8F, 9C13,F5     ;11251          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                    ;11252                                          ;SET UP TO CHECK TB DATA
U 0C90, 3022,15     ;11253          MOV MEM.DATA TO WR[0]           ;GET TB ADDRESS 0 DATA
U 0C91, 0869,3C     ;11254          JSR [CHECK.RESULT]              ;CHECK FOR 1'S
U 0C92, 08C8,54     ;11255          JMP [LOOP.T13.1]                ;ERROR LOOP IF ENABLED
                    ;11256
U 0C93, 369E,95     ;11257          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 0C94, FF12,15     ;11258          INC LS[9]                       ;TB ADDRESS 1
U 0C95, 9C13,F5     ;11259          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                    ;11260                                          ;SET UP TO CHECK TB DATA
```

```
U 0C96, 3022,15 ;11261          MOV MEM.DATA TO WR[0]            ;GET TB ADDRESS 1 DATA
U 0C97, 0869,3C ;11262          JSR [CHECK.RESULT]               ;CHECK FOR 1'S
U 0C98, 08C8,54 ;11263          JMP [LOOP.T13.1]                 ;ERROR LOOP IF ENABLED
                ;11264
U 0C99, FF82,15 ;11265          INC LS[ERROR.NUMBER]             ;ERROR 2
U 0C9A, B670,15 ;11266          MOV LS[OTHER.DATA] TO WR[0]      ;SET UP TO PRINT ADDRESS UNDER 'OTHER'
U 0C9B, 3E80,15 ;11267          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 24 IN CONTROL WORD.  CONSOLE
                ;11268                                           ; WILL PRINT UNDER 'OTHER' IF ERROR
                ;11269  LOOP.T13.2:
U 0C9C, 3E13,15 ;11270          MOV WR[2] TO LS[T9]              ;SET UP TB ADDRESS 0
U 0C9D, 6588,15 ;11271          CLR LS[ADDRESS.DATA]             ;ADDRESS TO PRINT IF ERROR
U 0C9E, 2F82,95 ;11272          CLR WR[1]                        ;EXPECTED DATA
U 0C9F, 1C12,F5 ;11273          MEM.REQ[WRITE.TB.STEP] ADRS[T9] DT[LONG]
                ;11274                                           ;SET UP TO WRITE 2 CONSECUTIVE TB LOC'S
U 0CA0, B29C,15 ;11275          WRITE.MEM LS[ZERO]               ;WRITE 0'S IN TB ADDRESS 0 AND 1
                ;11276
U 0CA1, 9C13,F5 ;11277          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                ;11278                                           ;SET UP TO CHECK TB ADDRESS 0
U 0CA2, 3022,15 ;11279          MOV MEM.DATA TO WR[0]            ;GET DATA
U 0CA3, 0869,3C ;11280          JSR [CHECK.RESULT]               ;CHECK FOR 0'S
U 0CA4, 88C9,C4 ;11281          JMP [LOOP.T13.2]                 ;ERROR LOOP IF ENABLED
                ;11282
U 0CA5, FF12,15 ;11283          INC LS[T9]                       ;SET TO ADDRESS 1
U 0CA6, FF88,15 ;11284          INC LS[ADDRESS.DATA]             ;ADDRESS TO PRINT IF ERROR (1)
U 0CA7, 2F82,95 ;11285          CLR WR[1]                        ;EXPECTED DATA
U 0CA8, 9C13,F5 ;11286          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                ;11287                                           ;SET UP TO CHECK TB ADDRESS 1
U 0CA9, 3022,15 ;11288          MOV MEM.DATA TO WR[0]            ;GET DATA
U 0CAA, 0869,3C ;11289          JSR [CHECK.RESULT]               ;CHECK FOR 0'S
U 0CAB, 88C9,C4 ;11290          JMP [LOOP.T13.2]                 ;ERROR LOOP IF ENABLED
                ;11291  END.T13:
```

```
;11292  .PAGE   "  TEST 14 TBA 09 and TB/UBS March Test (M8391  MCT Module)"
;11293  ;++
;11294  ;
;11295  ; Test Description:
;11296  ;
;11297  ;       The TBA 09 address line test checks that TBA 09 is high for UNIBUS map
;11298  ;       writes and low for TB writes.  The previous tests simply checked that
;11299  ;       this bit remained constant throughout the test.  If this bit fails, a
;11300  ;       UNIBUS write could go to the TB section of the RAM or vica-versa.  A
;11301  ;       march test will be run across all RAM addresses (i.e. TB RAM portion
;11302  ;       and CPU map portion) to check for any dual addressing.
;11303  ;         The data path for the TBA 09 test is the normal WRITE.TB and WRITE.
;11304  ;       UBS.MAP flows and their corresponding read flows.
;11305  ;
;11306  ; Assumptions:
;11307  ;
;11308  ;       It is assumed that all previous tests have run successfully.
;11309  ;
;11310  ; Test Steps:
;11311  ;
;11312  ;       1)  Set up error mask, error number, and module number in LS (for
;11313  ;           error printouts) and clear previous error number in LS.  The error
;11314  ;           mask will allow bits 1 through 22 to be checked.
;11315  ;       2)  Write a background of 0's in the RAM by writing all TB addresses
;11316  ;           and all UBS map addresses.
;11317  ;       3)  Read 0's, write 1's, read 1's in each TB location until all 128
;11318  ;           addresses have been accessed.
;11319  ;       4)  Continue read/write/read pattern in UBS map portion of RAM while
;11320  ;           incrementing the address until all 512 locations of UBS map have
;11321  ;           been accessed.
;11322  ;       5)  Starting at the top of UBS map RAM, read 1's, write 0's, read 0's
;11323  ;           while decrementing the address until all 512 locations have been
;11324  ;           accessed.
;11325  ;       6)  Continue read/write/read pattern in TB portion of RAM until all
;11326  ;           128 locations have been accessed.
;11327  ;
;11328  ; Errors:
;11329  ;
;11330  ;       NOTE:  OTHER data is TB or UBS map address.
;11331  ;
;11332  ;       Error 1 - March failed in TB portion of RAM, ascending addresses.
;11333  ;       Error 2 - March failed in UBS map portion of RAM, ascending addresses.
;11334  ;       Error 3 - March failed in UBS map portion of RAM, descending addresses.
;11335  ;       Error 4 - March failed in TB portion of RAM, descending addresses.
;11336  ;
;11337  ;
;11338  ; Debug:
;11339  ;
;11340  ;       Two methods of debug will be explained.  If the MCT module is in a
;11341  ;       test station then the MCT can be single stepped making debug easier
;11342  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;11343  ;       method will be explained also.  When single stepping the memory con-
;11344  ;       troller will help in debug, this section will explicitly suggest it.
;11345  ;
;11346  ;       Error 1 through 4 - The first area to check if this test fails is TBA
```

```
                    ;11347  ;       09 H address line.  This is especially true if all bits are failing or
                    ;11348  ;       a group of 4 bits is failing.  This can be checked while looping on
                    ;11349  ;       error.  It should be high on UBS map accesses and low on TB accesses.
                    ;11350  ;       If a group of 4 bits is failing, check it at the chip which outputs the
                    ;11351  ;       failing bit.  If incorrect, check the +3VB and ground connections on
                    ;11352  ;       the C portion of the MUX that outputs TBA 09 H.  If these connections
                    ;11353  ;       are OK, the MUX is probably bad.
                    ;11354  ;         The other possibility is a RAM failure internal to the address logic
                    ;11355  ;       in the chip.  Repalce the RAM chip which outputs the failing bit.
                    ;11356  ;
                    ;11357  ;--
                    ;11358
                    ;11359  T.14:
U 0CAC, B65E,15     ;11360          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;11361                                          ; STATUS WORD
U 0CAD, 3E80,15     ;11362          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;11363                                          ; BIT 15 INDICATES START OF TEST TO
                    ;11364                                          ; CONSOLE
U 0CAE, 10E0,15     ;11365          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;11366  WAIT.T14.0:
U 0CAF, 88CA,F4     ;11367          JMP [WAIT.T14.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;11368                                          ; RESPOND
                    ;11369
U 0CB0, 0A1A,AC     ;11370          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;11371
U 0CB1, B640,15     ;11372          MOV LS[#1] TO WR[0]              ;1 IN WR
U 0CB2, BEFC,15     ;11373          MOV WR[0] TO LS[SIZE]            ;SET UP SIZE REG FOR DATA TYPE OF WORD
U 0CB3, B62A,15     ;11374          MOV LS[#FF000000] TO WR[0]       ;SET TOP BYTE FOR ERROR MASK
U 0CB4, C76E,15     ;11375          BIS LS[BIT23] TO WR[0]           ;ALSO SET BIT 23
U 0CB5, C740,15     ;11376          BIS LS[BIT0] TO WR[0]            ;AND BIT 0
U 0CB6, 3E8A,15     ;11377          MOV WR[0] TO LS[ERRC   ASK]      ;WILL CHECK BITS 1 THROUGH 22
U 0CB7, B670,15     ;11378          MOV LS[OTHER.DATA] TO WR[0]      ;SET UP TO PRINT ADDRESS UNDER 'OTHER'
U 0CB8, 3E80,15     ;11379          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 24 IN CONTROL WORD TO FORCE
                    ;11380                                          ; PRINT UNDER 'OTHER' IF ERROR
                    ;11381
U 0CB9, 6512,15     ;11382          CLR LS[T9]                      ;START AT ADDRESS 0
                    ;11383  BACK.T14.1:
U 0CBA, 9812,F5     ;11384          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                    ;11385                                          ;SET UP TO WRITE IN TB PORTION OF RAM
U 0CBB, B29C,15     ;11386          WRITE.MEM LS[ZERO]              ;WRITE 0'S AT TB ADDRESS
U 0CBC, 0A18,2C     ;11387          JSR [NEXT.TB.ADDRESS]           ;GET NEXT TB ADDRESS
U 0CBD, 08CB,A4     ;11388          JMP [BACK.T14.1]                ;REPEAT IF COME BACK HERE (NOT DONE YET)
                    ;11389                                          ; ELSE WRITE BACKGROUND IN UB PORTION
U 0CBE, 6512,15     ;11390          CLR LS[T9]                      ;START AT ADDRESS 200 IN RAM (BIT 9 IS
                    ;11391                                          ; FORCED BY HARDWARE)
                    ;11392  BACK.T14.1A:
U 0CBF, 1B13,F5     ;11393          MEM.REQ[WRITE.UBS.MAP] ADRS[T9] DT[LONG]
                    ;11394                                          ;SET UP TO WRITE IN UB PORTION OF RAM
U 0CC0, B29C,15     ;11395          WRITE.MEM LS[ZERO]              ;WRITE 0'S AT UB ADDRESS
U 0CC1, 8A18,CC     ;11396          JSR [NEXT.UB.ADDRESS]           ;GET NEXT UB ADDRESS
U 0CC2, 08CB,F4     ;11397          JMP [BACK.T14.1A]               ;REPEAT IF COME BACK HERE (NOT DONE YET)
                    ;11398
U 0CC3, 6512,15     ;11399          CLR LS[T9]                      ;START AT ADDRESS 0
U 0CC4, 6588,15     ;11400          CLR LS[ADDRESS.DATA]            ;ADDRESS TO PRINT IF ERROR
                    ;11401  LOOP.T14.1:
```

```
U 0CC5, 2F82,95 ;11402          CLR WR[1]                           ;EXPECTED DATA
U 0CC6, 9C13,F5 ;11403          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                ;11404                                              ;SET UP TO READ TB
U 0CC7, 3022,15 ;11405          MOV MEM.DATA TO WR[0]               ;GET RESULT FROM TB
U 0CC8, 0869,3C ;11406          JSR [CHECK.RESULT]                  ;CHECK RESULT FOR ZEROS
U 0CC9, 88CC,54 ;11407          JMP [LOOP.T14.1]                    ;ERROR LOOP IF ENABLED
                ;11408
U 0CCA, 369E,95 ;11409          MOV LS[ONES] TO WR[1]               ;EXPECTED DATA
U 0CCB, 9812,F5 ;11410          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                ;11411                                              ;SET UP TO WRITE 1'S
U 0CCC, 329E,15 ;11412          WRITE.MEM LS[ONES]                  ;WRITE 1'S IN TB
U 0CCD, 9C13,F5 ;11413          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                ;11414                                              ;SET UP TO READ RESULT
U 0CCE, 3022,15 ;11415          MOV MEM.DATA TO WR[0]               ;READ TB DATA
U 0CCF, 0869,3C ;11416          JSR [CHECK.RESULT]                  ;CHECK RESULT FOR ONES
U 0CD0, 88CC,54 ;11417          JMP [LOOP.T14.1]                    ;ERROR LOOP IF ENABLED
                ;11418
U 0CD1, 0A18,2C ;11419          JSR [NEXT.TB.ADDRESS]               ;INCREMENT ADDRESS
U 0CD2, 88CC,54 ;11420          JMP [LOOP.T14.1]                    ;REPEAT IF NOT DONE
                ;11421                                              ; ELSE SUBROUTINE RETURNS TO NEXT ADDR
                ;11422                                              ; (CHECK UB PORTION)
                ;11423
U 0CD3, FF82,15 ;11424          INC LS[ERROR.NUMBER]                ;ERROR 2
U 0CD4, 6512,15 ;11425          CLR LS[T9]                          ;START AT ADDRESS 0 (HARDWARE FORCES
                ;11426                                              ; RAM BIT 9 FOR ADDRESS 200)
U 0CD5, B652,15 ;11427          MOV LS[#200] TO WR[0]               ;SET UP ADDRESS TO PRINT
U 0CD6, BE88,15 ;11428          MOV WR[0] TO LS[ADDRESS.DATA]       ;ADDRESS TO PRINT IF ERROR (200 IS
                ;11429                                              ; FIRST RAM ADDRESS IN UB)
                ;11430  LOOP.T14.2:
U 0CD7, 2F82,95 ;11431          CLR WR[1]                           ;EXPECTED DATA
U 0CD8, 1C13,75 ;11432          MEM.REQ[READ.UBS.MAP] ADRS[T9] DT[LONG]
                ;11433                                              ;SET UP TO READ UB
U 0CD9, 3022,15 ;11434          MOV MEM.DATA TO WR[0]               ;GET RESULT FROM UB
U 0CDA, 0869,3C ;11435          JSR [CHECK.RESULT]                  ;CHECK RESULT FOR ZEROS
U 0CDB, 88CD,74 ;11436          JMP [LOOP.T14.2]                    ;ERROR LOOP IF ENABLED
                ;11437
U 0CDC, 369E,95 ;11438          MOV LS[ONES] TO WR[1]               ;EXPECTED DATA
U 0CDD, 1B13,F5 ;11439          MEM.REQ[WRITE.UBS.MAP] ADRS[T9] DT[LONG]
                ;11440                                              ;SET UP TO WRITE 1'S
U 0CDE, 329E,15 ;11441          WRITE.MEM LS[ONES]                  ;WRITE 1'S IN UB
U 0CDF, 1C13,75 ;11442          MEM.REQ[READ.UBS.MAP] ADRS[T9] DT[LONG]
                ;11443                                              ;SET UP TO READ RESULT
U 0CE0, 3022,15 ;11444          MOV MEM.DATA TO WR[0]               ;READ UB DATA
U 0CE1, 0869,3C ;11445          JSR [CHECK.RESULT]                  ;CHECK RESULT FOR ONES
U 0CE2, 88CD,74 ;11446          JMP [LOOP.T14.2]                    ;ERROR LOOP IF ENABLED
                ;11447
U 0CE3, 8A18,CC ;11448          JSR [NEXT.UB.ADDRESS]               ;INCREMENT ADDRESS
U 0CE4, 88CD,74 ;11449          JMP [LOOP.T14.2]                    ;REPEAT IF NOT DONE
                ;11450                                              ; ELSE SUBROUTINE RETURNS TO NEXT ADDR
                ;11451
U 0CE5, FF82,15 ;11452          INC LS[ERROR.NUMBER]                ;ERROR 3
U 0CE6, 6512,15 ;11453          CLR LS[T9]                          ;START AT ADDRESS 0 (HARDWARE FORCES
                ;11454                                              ; RAM BIT 9 FOR ADDRESS 200)
U 0CE7, B654,15 ;11455          MOV LS[#400] TO WR[0]               ;SET UP ADDRESS TO PRINT
U 0CE8, 2100,15 ;11456          DEC WR[0]                           ;NOW HAVE 3FF(H) AS ADDRESS
```

```
U 0CE9, BE88,15 ;11457          MOV WR[0] TO LS[ADDRESS.DATA]   ;ADDRESS TO PRINT IF ERROR
                ;11458
                ;11459  REPEAT.T14.3:
U 0CEA, 7D12,15 ;11460          COM LS[T9]                      ;NOW DECREMENTING ADDRESS IN UB
                ;11461  LOOP.T14.3:
U 0CEB, 369E,95 ;11462          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 0CEC, 1C13,75 ;11463          MEM.REQ[READ.UBS.MAP] ADRS[T9] DT[LONG]
                ;11464                                          ;SET UP TO READ UB
U 0CED, 3022,15 ;11465          MOV MEM.DATA TO WR[0]           ;GET RESULT FROM UB
U 0CEE, 0869,3C ;11466          JSR [CHECK.RESULT]              ;CHECK RESULT FOR ZEROS
U 0CEF, 88CE,B4 ;11467          JMP [LOOP.T14.3]                ;ERROR LOOP IF ENABLED
                ;11468
U 0CF0, 2F82,95 ;11469          CLR WR[1]                       ;EXPECTED DATA
U 0CF1, 1B13,F5 ;11470          MEM.REQ[WRITE.UBS.MAP] ADRS[T9] DT[LONG]
                ;11471                                          ;SET UP TO WRITE 0'S
U 0CF2, B29C,15 ;11472          WRITE.MEM LS[ZERO]              ;WRITE 0'S IN UB
U 0CF3, 1C13,75 ;11473          MEM.REQ[READ.UBS.MAP] ADRS[T9] DT[LONG]
                ;11474                                          ;SET UP TO READ RESULT
U 0CF4, 3022,15 ;11475          MOV MEM.DATA TO WR[0]           ;READ UB DATA
U 0CF5, 0869,3C ;11476          JSR [CHECK.RESULT]              ;CHECK RESULT FOR ONES
U 0CF6, 88CE,B4 ;11477          JMP [LOOP.T14.3]                ;ERROR LOOP IF ENABLED
                ;11478
U 0CF7, 7D12,15 ;11479          COM LS[T9]                      ;RECOMPLEMENT ADDRESS
U 0CF8, FC88,15 ;11480          DEC LS[ADDRESS.DATA]            ;DECREMENT ADDRESS TO PRINT
U 0CF9, FC88,15 ;11481          DEC LS[ADDRESS.DATA]            ;TWICE SINCE SUBROUTINE INCREMENTS IT
U 0CFA, 8A18,CC ;11482          JSR [NEXT.UB.ADDRESS]           ;INCREMENT ADDRESS
U 0CFB, 08CE,A4 ;11483          JMP [REPEAT.T14.3]              ;REPEAT IF HERE UNTIL DONE
                ;11484                                          ; ELSE SUBROUTINE RETURNS TO NEXT ADDR
                ;11485
U 0CFC, FF82,15 ;11486          INC LS[ERROR.NUMBER]            ;ERROR 4
U 0CFD, 6512,15 ;11487          CLR LS[T9]                      ;START AT ADDRESS 0
U 0CFE, 364E,15 ;11488          MOV LS[#80] TO WR[0]            ;SET UP ADDRESS TO PRINT
U 0CFF, 2100,15 ;11489          DEC WR[0]                       ;NOW HAVE TOP ADDRESS (7F) IN TB
U 0D00, BE88,15 ;11490          MOV WR[0] TO LS[ADDRESS.DATA]   ;ADDRESS TO PRINT IF ERROR
                ;11491  REPEAT.T14.4:
U 0D01, 7D12,15 ;11492          COM LS[T9]                      ;NOW DECREMENTING ADDRESS IN UB
                ;11493  LOOP.T14.4:
U 0D02, 369E,95 ;11494          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 0D03, 9C13,F5 ;11495          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                ;11496                                          ;SET UP TO READ TB
U 0D04, 3022,15 ;11497          MOV MEM.DATA TO WR[0]           ;GET RESULT FROM TB
U 0D05, 0869,3C ;11498          JSR [CHECK.RESULT]              ;CHECK RESULT FOR ZEROS
U 0D06, 88D0,24 ;11499          JMP [LOOP.T14.4]                ;ERROR LOOP IF ENABLED
                ;11500
U 0D07, 2F82,95 ;11501          CLR WR[1]                       ;EXPECTED DATA
U 0D08, 9812,F5 ;11502          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                ;11503                                          ;SET UP TO WRITE 1'S
U 0D09, B29C,15 ;11504          WRITE.MEM LS[ZERO]              ;WRITE 0'S IN TB
U 0D0A, 9C13,F5 ;11505          MEM.REQ[READ.TB] ADRS[T9] DT[LONG]
                ;11506                                          ;SET UP TO READ RESULT
U 0D0B, 3022,15 ;11507          MOV MEM.DATA TO WR[0]           ;READ TB DATA
U 0D0C, 0869,3C ;11508          JSR [CHECK.RESULT]              ;CHECK RESULT FOR ONES
U 0D0D, 88D0,24 ;11509          JMP [LOOP.T14.4]                ;ERROR LOOP IF ENABLED
                ;11510
U 0D0E, 7D12,15 ;11511          COM LS[T9]                      ;RECOMPLEMENT ADDRESS
```

```
U 0D0F, FC88,15 ;11512          DEC LS[ADDRESS.DATA]          ;DECREMENT ADDRESS TO PRINT
U 0D10, FC88,15 ;11513          DEC LS[ADDRESS.DATA]          ;TWICE SINCE SUBROUTINE INCREMENTS IT
U 0D11, 0A18,2C ;11514          JSR [NEXT.TB.ADDRESS]         ;INCREMENT ADDRESS
U 0D12, 88D0,14 ;11515          JMP [REPEAT.T14.4]            ;REPEAT IF HERE UNTIL DONE
                ;11516                                        ; ELSE SUBROUTINE RETURNS TO NEXT ADDR
                ;11517                                        ; (DONE)
                ;11518  END.T14:
```

```
;11519  .PAGE   "  TEST 15 TB Miss and Tag Field Test (M8391  MCT Module)"
;11520  ;++
;11521  ;
;11522  ; Test Description:
;11523  ;
;11524  ;       This test checks the Tag logic including the 256X4 RAMS, the compar-
;11525  ;       ators that produce TB miss, and the CSR 1 logic for the TB miss bit
;11526  ;       (bit 21).  The test will issue a WRITE.TB to load the TB and tag fields
;11527  ;       and then issue a TEST.V.RCHK to clock the CSR with the resulting TB
;11528  ;       MISS.  The TB miss logic will be checked first.  When this has been
;11529  ;       verified, the tag field will be tested.  The last part of this test
;11530  ;       will check that a WRITE.UBS.MAP is prevented from modifying the con-
;11531  ;       tents of the tag field RAM and that BYTE OFFSET clear will enable a
;11532  ;       TB MISS regardless of a comparison.
;11533  ;         Data for the tag field comes from LVA 15 through 30 on the write to
;11534  ;       TB.  This is loaded into RAM at an address specified by LVA 9 through
;11535  ;       14 and LVA 31 (128 decimal locations).  RAM address bit 7 is forced
;11536  ;       low by hardware.  Write to the tag RAMS is inhibited on a write to the
;11537  ;       UNIBUS map.  The tag bits are compared to LVA 15 through 30 sent over
;11538  ;       with the TEST.V.RCHK command to produce TB MISS L.  This bit is clocked
;11539  ;       into CSR 1 bit 21 if BYTE OFFSET is set.
;11540  ;         The data path for the WRITE.TB is described under the TB RAM test.
;11541  ;       The data path for the TEST.V.RCHK is as follows:  A MEM.REQ is issued
;11542  ;       with MF=TEST.V.RCHK (2H) and the memory dispatches to the proper rou-
;11543  ;       tine as described in the beginning of this listing.  At the dispatch
;11544  ;       address, a rotate clock is issued (this is used to set up OP ERR, but
;11545  ;       this is not tested in this test), the VAR BYPASS is enabled, and memory
;11546  ;       busy is asserted.  The following cycle simply asserts VAR BYPASS and
;11547  ;       memory busy.  The next cycle clocks CSR 1 with any error bits that
;11548  ;       are set and clears the RDS and CRD bits.  The MCT transceivers are en-
;11549  ;       abled so that the PA bits from the TB and some current VAR bits can be
;11550  ;       read.  VAR bypass and memory busy are still asserted.  The following
;11551  ;       cycle repeats this except VAR bypass is dropped.  The next cycle loops
;11552  ;       waiting for CPU grant to go away (by the CPU reading the data on the MC
;11553  ;       BUS via a MOV MEM.DATA TO WR instruction) while keeping the MCT trans-
;11554  ;       ceivers enabled.  After CPU grant is dropped, preparation is made for a
;11555  ;       UNIBUS cycle and the micro-code returns to IDLE.
;11556  ;         Data patterns used for this test are all 1's, all 0's, shifting 1's,
;11557  ;       and shifting 0's for the TB miss logic test and TAG RAM test.  Also a
;11558  ;       moving inversion test on the TAG RAMS will be performed.
;11559  ;
;11560  ; Assumptions:
;11561  ;
;11562  ;       It is assumed that all previous tests have run successfully.
;11563  ;
;11564  ; Test Steps:
;11565  ;
;11566  ;       1)  Set up error mask, error number, and module number in LS (for
;11567  ;           error printouts) and clear previous error number in LS.  Also set
;11568  ;           MME bit in CSR 1.
;11569  ;       2)  Clear WR 1 and issue MEM.REQ with MF=WRITE.TB and DT=longword.  Use
;11570  ;           an LS location containing 0 for the address.
;11571  ;       3)  Issue WRITE.MEM LS using an LS location containing a 1 in bit 25
;11572  ;           (writes to byte offset), others 0.  Now tag field contains 0's.
;11573  ;       4)  Issue MEM.REQ with MF=TEST.V.RCHK  and DT=longword.  Use an LS loc-
```

```
;11574  ;           ation containing 0's as the address.  Then issue MOV MEM.DATA to
;11575  ;           WR[0] and check for 0's (checking PA bits from TB and LVA bits sent
;11576  ;           over is mainly a micro-code check).
;11577  ;       5)  Change error mask to check bit 21 (TB MISS) only and issue READ.CSR
;11578  ;           with 1 in LVA 02.  Bit 21 should be clear.
;11579  ;       6)  Issue MEM.REQ with MF=TEST.V.RCHK  and DT=longword.  Use an LS loc-
;11580  ;           ation containing shifting 1's in bits 15 to 30 as the address. Then
;11581  ;           issue MOV MEM.DATA to WR[0] (no need to check this).  Issue READ.CSR
;11582  ;           with 1 in LVA 02 and check for TB MISS (bit 21) set.
;11583  ;       7)  Repeat step 6 until bits 15 to 30 have each been 1.
;11584  ;       8)  Set error mask up to check bits 0-22 and 24-31, set WR 1, and issue
;11585  ;           MEM.REQ with MF=WRITE.TB and DT=longword.  Use an LS location con-
;11586  ;           taining 0 for the address.
;11587  ;       9)  Issue WRITE.MEM LS using an LS location containing all 1's.  Now
;11588  ;           tag field contains 1's.
;11589  ;       10) Issue MEM.REQ with MF=TEST.V.RCHK  and DT=longword.  Use an LS loc-
;11590  ;           ation containing 1's in bits 15 to 30 as the address.  Then issue
;11591  ;           MOV MEM.DATA to WR[0] and check for 1's.
;11592  ;       11) Change error mask to check bit 21 (TB MISS) only and issue READ.CSR
;11593  ;           with 1 in LVA 02.  Bit 21 should be clear.
;11594  ;       12) Issue MEM.REQ with MF=TEST.V.RCHK  and DT=longword.  Use an LS loc-
;11595  ;           ation containing shifting 0's in bits 15 to 30 as the address. Then
;11596  ;           issue MOV MEM.DATA to WR[0] (no need to check this).  Issue READ.CSR
;11597  ;           with 1 in LVA 02 and check for TB MISS (bit 21) set.
;11598  ;       13) Repeat step 6 until bits 15 to 30 have each been 0.
;11599  ;       14) Write TB with shifting 1's in tag field at address 0.  Issue TEST.V
;11600  ;           .RCHK with shifting 1's in LVA 15-30 and check CSR 1 TB MISS for 0.
;11601  ;       15) Repeat step 14 on all other addresses.
;11602  ;       16) Repeat steps 14 and 15 using shifting 0's as data.
;11603  ;       17) Execute march type test with shifting tag RAM address bits to test
;11604  ;           for address line faults.
;11605  ;       18) Execute moving inversion test on tag RAM.
;11606  ;       19) Write 0's in tag field, write 1's in UNIBUS MAP, read tag field and
;11607  ;           check for no change (0's).
;11608  ;       20) Write 0's in tag field and BYTE OFFSET bit.  Issue TEST.V.RCHK with
;11609  ;           match in LVA 15-30.  Check that BYTE OFFSET clear forces a TB miss.
;11610  ;
;11611  ; Errors:
;11612  ;
;11613  ;       NOTE:  Expected and received data is TB mis bit in CSR 1.
;11614  ;
;11615  ;       Error 1 - TEST.V.RCHK failed to return correct PA bits or LVA bits.
;11616  ;                 Possible micro-store PROM failure.  RAM address under OTHER
;11617  ;                 in error report.
;11618  ;       NOTE:  In errors 2 and 3 below, OTHER data will be LVA bits 15 through
;11619  ;              30 that is compared with all 0's in the TAG RAM.
;11620  ;       Error 2 - TB MISS set on match of all 0's.  See note above.
;11621  ;       Error 3 - TB MISS failed to set on mismatch of 0's and shifting 1's.
;11622  ;                 See note above.
;11623  ;       Error 4 - TEST.V.RCHK failed to return correct PA bits or LVA bits.
;11624  ;                 Possible micro-store PROM failure.  RAM address under OTHER
;11625  ;                 in error report.
;11626  ;       NOTE:  In errors 5 and 6 below, OTHER data will be LVA bits 15 through
;11627  ;              30 that is compared with all 1's in the TAG RAM.
;11628  ;       Error 5 - TB MISS set on match of all 1's.  See note above.
```

```
;11629  ;       Error 6 - TB MISS failed to set on mismatch of 1's and shifting 0's.
;11630  ;                 See note above.
;11631  ;       Error 7 - Tag field RAM failed shifting 1's test at RAM address under
;11632  ;                 'OTHER DATA'.  Failing shift pattern is in WR 2.
;11633  ;       Error 8 - Tag field RAM failed shifting 0's test at RAM address under
;11634  ;                 'OTHER DATA'.  Failing shift pattern is in WR 2.
;11635  ;       Error 9 - Tag field RAM failed march test with shifting address bits.
;11636  ;                 Address under OTHER in error report.
;11637  ;       Error A - Tag field RAM failed moving inversion test.  Address is in
;11638  ;                 LS 9.
;11639  ;       Error B - Tag field RAM written on write to UNIBUS map (write disable
;11640  ;                 failed).  OTHER data is tag RAM address.
;11641  ;       Error C - TB MISS failed to be asserted on match with BYTE OFFSET
;11642  ;                 clear.  OTHER data is tag RAM address.
;11643  ;
;11644  ; Debug:
;11645  ;
;11646  ;       ***NOTE***When the data under OTHER in the error report is an address,
;11647  ;       the bits will be arranged so as to give the actual RAM address in
;11648  ;       hexadecimal format.
;11649  ;
;11650  ;       Two methods of debug will be explained.  If the MCT module is in a
;11651  ;       test station then the MCT can be single stepped making debug easier
;11652  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;11653  ;       method will be explained also.  When single stepping the memory con-
;11654  ;       troller will help in debug, this section will explicitly suggest it.
;11655  ;
;11656  ;       Error 1 - This error indicates a possible problem with the control
;11657  ;       store PROMS.  All the logic used to write and read the data has been
;11658  ;       checked in previous tests.  If MCT single step is available, the bits
;11659  ;       can be checked on each cycle for errors.
;11660  ;
;11661  ;       Error 2 - This error indicates that TB MISS was asserted when a match
;11662  ;       of all 0's was executed.  Either the TAG field RAM, tag comparators, or
;11663  ;       CSR 1 logic failed.  Single step the CPU to the MOVE MEM.DATA TO WR[0]
;11664  ;       instruction after the TEST.V.RCHK memory request.  The MCT will loop
;11665  ;       waiting for CPU DATA RCVD to clear CPU GRANT.  At this time TB MISS L
;11666  ;       can be checked for a high.  If TB MISS L is low, check the inputs to
;11667  ;       the comparators.  All the TAG bits and LVA bits should be low.  If they
;11668  ;       are, the comparator is bad.  If the TAG bits are bad, check pin 20 and
;11669  ;       19 of the tag field RAMS for floats.  The signals themselves were
;11670  ;       checked previously.  Suspect a bad RAM or bad LVA input if there are no
;11671  ;       floats.
;11672  ;         If TB MISS L is high, check it at the input to the CSR 1A PAL.  Also
;11673  ;       check BYTE OFFSET H for a high.  If these are both OK, the CSR 1A PAL
;11674  ;       is the most likely suspect.
;11675  ;
;11676  ;       Error 3 - This error indicates that TB MISS was not asserted when a
;11677  ;       mismatch of one bit occurred.  Either the TAG field RAM, tag compara-
;11678  ;       tors, or CSR 1 logic failed.  Single step the CPU to the MOVE MEM.DAT
;11679  ;       TO WR[0] instruction after the TEST.V.RCHK memory request.  The MCT
;11680  ;       will loop waiting for CPU DATA RCVD to clear CPU GRANT.  At this time
;11681  ;       TB MISS L can be checked for a low.  If TB MISS L is high, check the
;11682  ;       inputs to the comparators.  All the TAG bits should be low, and all the
;11683  ;       LVA bits except one should be low.  If they are, the comparator is bad.
```

```
                ;11684  ;       If TB MISS L is low, check it at the input to the CSR 1A PAL.  If
                ;11685  ;     this is OK, the CSR 1A PAL is the most likely suspect.
                ;11686  ;
                ;11687  ;     Error 4 - See error 1.
                ;11688  ;
                ;11689  ;     Error 5 - See error 2.  The only difference is that all inputs to the
                ;11690  ;     comparators should be high (1's).
                ;11691  ;
                ;11692  ;     Error 6 - See error 3.  The only difference is that all inputs to the
                ;11693  ;     comparators should be high on the TAG bits and on all but 1 of the LVA
                ;11694  ;     bits.
                ;11695  ;
                ;11696  ;     Error 7 - This error is most likely a RAM failure.  Single step the
                ;11697  ;     CPU to the MOV MEM.DATA TO WR[0] instruction which follows the MEM.REQ
                ;11698  ;     that does the TEST.V.RCHK.  The MCT will be looping waiting for CPU
                ;11699  ;     GRANT to be cleared.  The output of the RAMs can be checked at this
                ;11700  ;     time to find the failing bit.
                ;11701  ;
                ;11702  ;     Error 8 - See error 7.
                ;11703  ;
                ;11704  ;     Error 9 - This error indicates a problem with the address lines or an
                ;11705  ;     internal address decode problem in the RAM.  The address lines can be
                ;11706  ;     checked by setting the SOMM to the MOV MEM.DATA TO WR[0] instruction
                ;11707  ;     that follows the TEST.V.RCHK.  As the CPU halts on the micro-match, the
                ;11708  ;     address lines to the RAM can be checked for a shifting 1 across a field
                ;11709  ;     of 0's.  After each CONTINUE command is issued from the terminal, the
                ;11710  ;     address bit will shift.  If the address lines themselves are OK, sus-
                ;11711  ;     pect the RAM.
                ;11712  ;
                ;11713  ;     Error A - Most likely a bad RAM would produce this error.  Follow
                ;11714  ;     error 9 above to pinpoint the failing RAM.
                ;11715  ;
                ;11716  ;     Error B - This error indicates a problem with either the input TBA 9 H
                ;11717  ;     or the tag RAM itself. Single step the CPU to the MEM.REQ WRITE.UBS.MAP
                ;11718  ;     instruction.  The input TBA 9 H to the tag field RAMs can be checked
                ;11719  ;     for a high at each of the tag field RAMs at that time.  If these inputs
                ;11720  ;     are OK, then then one of the RAMs is bad.
                ;11721  ;       To determine the failing RAM, single step the CPU to the MOV MEM.DAT
                ;11722  ;     TO WR[0] instruction that follows the TEST.V.RCHK and look at the out-
                ;11723  ;     put of the RAMs.  The RAM outputing a 1 is the bad one.
                ;11724  ;
                ;11725  ;     Error C - This error indicates a problem with the CSR 1A PAL or the
                ;11726  ;     BYTE OFFSET input.  Single step the CPU to the MOV MEM.DATA TO WR[0]
                ;11727  ;     instruction following the TEST.V.RCHK, and check the input to the CSR
                ;11728  ;     1A PAL for a low on BYTE OFFSET.  If this is correct, the CSR 1A PAL is
                ;11729  ;     probably bad.
                ;11730  ;
                ;11731  ;--
                ;11732
                ;11733  T.15:
U 0D13, B65E,15 ;11734          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;11735                                          ; STATUS WORD
U 0D14, 3E80,15 ;11736          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;11737                                          ; BIT 15 INDICATES START OF TEST TO
                ;11738                                          ; CONSOLE
```

```
U 0D15, 10E0,15 ;11739          MISC [SET.CP.ATTN]                  ;ISSUE CPU ATTN TO CONSOLE
                ;11740  WAIT.T15.0:
U 0D16, 88D1,64 ;11741          JMP [WAIT.T15.0]                    ;LOOP HERE WAITING FOR CONSOLE TO
                ;11742                                              ; RESPOND
                ;11743
U 0D17, 0A1A,AC ;11744          JSR [SETUP.1]                       ;SET UP MASKS, MODULE CODE ETC.
                ;11745
U 0D18, B66E,15 ;11746          MOV LS[BIT23] TO WR[0]              ;MASK OUT BIT 23 (PA 24)
U 0D19, 3E8A,15 ;11747          MOV WR[0] TO LS[ERROR.MASK]         ;SET UP ERROR MASK TO CHECK ALL BITS
                ;11748                                              ; EXCEPT BIT 23
U 0D1A, B670,15 ;11749          MOV LS[OTHER.DATA] TO WR[0]         ;SET UP TO PRINT ADDRESS UNDER 'OTHER'
U 0D1B, 3E80,15 ;11750          MOV WR[0] TO LS[CONTROL.STATUS]     ;SET BIT 24 IN CONTROL WORD TO FORCE
                ;11751                                              ; PRINT OF ADDRESS UNDER OTHER IF ERROR
U 0D1C, 0A17,DC ;11752          JSR [WRITE.CSR1.MME]                ;ENABLE MEMORY MANAGEMANT
U 0D1D, B640,15 ;11753          MOV LS[#1] TO WR[0]                 ;1 IN WR
U 0D1E, BEFC,15 ;11754          MOV WR[0] TO LS[SIZE]               ;SET UP SIZE REG FOR DATA TYPE OF WORD
U 0D1F, 6588,15 ;11755          CLR LS[ADDRESS.DATA]                ;CLEAR ADDRESS THAT GETS PRINTED
U 0D20, 989C,F5 ;11756          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;11757                                              ;SET UP TO WRITE TB ADDRESS 0 WITH 0'S
                ;11758                                              ; IN TAG FIELD
U 0D21, B272,15 ;11759          WRITE.MEM LS[BIT25]                 ;WRITE TB WITH 1 IN BYTE OFFSET BIT,
                ;11760                                              ; 0'S IN OTHERS
                ;11761
                ;11762  LOOP.T15.1:
U 0D22, B640,15 ;11763          MOV LS[#1] TO WR[0]                 ;1 IN WR
U 0D23, BE82,15 ;11764          MOV WR[0] TO LS[ERROR.NUMBER]       ;SET UP ERROR NUMBER IN LS
U 0D24, 989D,75 ;11765          MEM.REQ[TEST.V.RCHK] ADRS[#0] DT[LONG]
                ;11766                                              ;SET UP TO CHECK TB AND CLOCK CSR 1
                ;11767                                              ; WITH TB MISS BIT
U 0D25, 3022,15 ;11768          MOV MEM.DATA TO WR[0]               ;READ PA AND LVA BITS FROM TB 0 (MAINLY
                ;11769                                              ; A MICRO-CODE CHECK)
U 0D26, 2F82,95 ;11770          CLR WR[1]                           ;EXPECTED DATA
U 0D27, 0869,3C ;11771          JSR [CHECK.RESULT]                  ;CHECK PA AND LVA BITS FOR ALL 0'S
U 0D28, 08D2,24 ;11772          JMP [LOOP.T15.1]                    ;ERROR LOOP IF ENABLED
                ;11773
U 0D29, FF82,15 ;11774          INC LS[ERROR.NUMBER]                ;ERROR 2
U 0D2A, 5F6A,15 ;11775          MCOM LS[TB.MISS] TO WR[0]           ;CLEAR BIT 21 IN WR 0, SET OTHERS
U 0D2B, 3E8A,15 ;11776          MOV WR[0] TO LS[ERROR.MASK]         ;SET UP TO CHECK BIT 21 ONLY (CSR 1
                ;11777                                              ; TB MISS BIT)
U 0D2C, 9D45,75 ;11778          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;11779                                              ;SET UP TO READ CSR 1
U 0D2D, 3022,15 ;11780          MOV MEM.DATA TO WR[0]               ;GET DATA FROM CSR 1
U 0D2E, 2F82,95 ;11781          CLR WR[1]                           ;EXPECTED DATA
U 0D2F, 0869,3C ;11782          JSR [CHECK.RESULT]                  ;CHECK FOR BIT 21 CLEAR (NO TB MISS)
U 0D30, 08D2,24 ;11783          JMP [LOOP.T15.1]                    ;ERROR LOOP IF ENABLED
                ;11784
U 0D31, FF82,15 ;11785          INC LS[ERROR.NUMBER]                ;ERROR 3
U 0D32, 365F,15 ;11786          MOV LS[BIT15] TO WR[2]              ;SET BIT 15 IN WR
                ;11787  REPEAT.T15.3:
U 0D33, 3E13,15 ;11788          MOV WR[2] TO LS[T9]                 ;SET UP ADDRESS IN LS 9 (SHIFTING 1
                ;11789                                              ; FROM LVA 15 THRU LVA 30)
U 0D34, 3E89,15 ;11790          MOV WR[2] TO LS[ADDRESS.DATA]       ;SET UP TO PRINT VALUE OF LVA 15 THRU
                ;11791                                              ; LVA 30 UNDER OTHER DATA
                ;11792  LOOP.T15.3:
U 0D35, B66A,95 ;11793          MOV LS[TB.MISS] TO WR[1]            ;EXPECTED DATA (TB MISS SET)
```

```
U 0D36, 9813,75 ;11794          MEM.REQ[TEST.V.RCHK] ADRS[T9] DT[LONG]
                ;11795                                  ;SET UP TO ISSUE TEST.V.RCHK WITH TAG
                ;11796                                  ; FIELD MISMATCH
U 0D37, 3022,15 ;11797          MOV MEM.DATA TO WR[0]    ;ISSUE MOV TO COMPLETE MEMORY CYCLE
                ;11798                                  ; (NO NEED TO CHECK THIS DATA)
U 0D38, 9D45,75 ;11799          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;11800                                  ;SET UP TO READ CSR 1
U 0D39, 3022,15 ;11801          MOV MEM.DATA TO WR[0]    ;GET DATA FROM CSR 1
U 0D3A, 0869,3C ;11802          JSR [CHECK.RESULT]       ;CHECK FOR TB MISS SET (BIT 21)
U 0D3B, 08D3,54 ;11803          JMP [LOOP.T15.3]         ;ERROR LOOP IF ENABLED
                ;11804
U 0D3C, 23C1,15 ;11805          ROL WR[2]                ;SHIFT BIT LEFT
                ;11806          TST WR[2],               ;SEE IF BIT 31 SET
U 0D3D, 2005,35 ;11807           DT(LONG)&SET.ALU.CC     ; AND SET CONDITION CODES
U 0D3E, 88D3,30 ;11808          JMP.IF[N.CLR] TO [REPEAT.T15.3] ;REPEAT IF BIT 31 HAS NOT SET YET
                ;11809                                  ; ELSE GO ON
                ;11810
U 0D3F, FF82,15 ;11811          INC LS[ERROR.NUMBER]     ;ERROR 4
U 0D40, 3683,95 ;11812          MOV LS[ERROR.NUMBER] TO WR[3]   ;SAVE ERROR NUMBER
U 0D41, B736,15 ;11813          MOV LS[#7FFF8000] TO WR[0]      ;GET DATA FOR TB ADDRESS
U 0D42, C726,15 ;11814          BIS LS[#FF] TO WR[0]            ;SET LVA BITS 0-7 FOR TEST.V.RCHK
U 0D43, 4750,15 ;11815          BIS LS[BIT8] TO WR[0]           ;AND ALSO SET LVA 8
U 0D44, BE12,15 ;11816          MOV WR[0] TO LS[T9]             ;STORE IN LS 9
U 0D45, 9812,F5 ;11817          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                ;11818                                  ;SET UP TO WRITE TB 0 WITH 1'S IN TAG
                ;11819                                  ; FIELD
U 0D46, 329E,15 ;11820          WRITE.MEM LS[ONES]       ;WRITE ALL ONES IN TB ADDRESS 0
                ;11821  LOOP.T15.4:
U 0D47, 6588,15 ;11822          CLR LS[ADDRESS.DATA]     ;CLEAR ADDRESS TO PRINT
U 0D48, BE83,95 ;11823          MOV WR[3] TO LS[ERROR.NUMBER]   ;ERROR NUMBER 4
U 0D49, B66E,15 ;11824          MOV LS[BIT23] TO WR[0]          ;MASK OUT BIT 23 (PA 24)
U 0D4A, 3E8A,15 ;11825          MOV WR[0] TO LS[ERROR.MASK]     ;SET UP ERROR MASK TO CHECK ALL BITS
                ;11826                                  ; EXCEPT BIT 23
U 0D4B, 9813,75 ;11827          MEM.REQ[TEST.V.RCHK] ADRS[T9] DT[LONG]
                ;11828                                  ;SET UP TO CHECK TB AND CLOCK CSR 1
                ;11829                                  ; WITH TB MISS BIT
U 0D4C, 3022,15 ;11830          MOV MEM.DATA TO WR[0]    ;READ PA AND LVA BITS FROM TB 0 (MAINLY
                ;11831                                  ; A MICRO-CODE CHECK)
U 0D4D, 5F7C,95 ;11832          MCOM LS[BIT30] TO WR[1]  ;EXPECTED DATA (ALL BITS SET EXCEPT LVA
                ;11833                                  ; 31 WHICH COMES BACK AS BIT 30)
U 0D4E, 0869,3C ;11834          JSR [CHECK.RESULT]       ;CHECK PA AND LVA BITS FOR ALL 1'S
U 0D4F, 08D4,74 ;11835          JMP [LOOP.T15.4]         ;ERROR LOOP IF ENABLED
                ;11836
U 0D50, FF82,15 ;11837          INC LS[ERROR.NUMBER]     ;ERROR 5
U 0D51, B736,15 ;11838          MOV LS[#7FFF8000] TO WR[0]      ;LVA 15 THROUGH 30 SET
U 0D52, BE88,15 ;11839          MOV WR[0] TO LS[ADDRESS.DATA]   ;OTHER DATA IS LVA 15 THROUGH 30
U 0D53, 5F6A,15 ;11840          MCOM LS[TB.MISS] TO WR[0]       ;CLEAR BIT 21 IN WR 0, SET OTHERS
U 0D54, 3E8A,15 ;11841          MOV WR[0] TO LS[ERROR.MASK]     ;SET UP TO CHECK BIT 21 ONLY (CSR 1
                ;11842                                  ; TB MISS BIT)
U 0D55, 9D45,75 ;11843          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;11844                                  ;SET UP TO READ CSR 1
U 0D56, 3022,15 ;11845          MOV MEM.DATA TO WR[0]    ;GET DATA FROM CSR 1
U 0D57, 2F82,95 ;11846          CLR WR[1]                ;EXPECTED DATA
U 0D58, 0869,3C ;11847          JSR [CHECK.RESULT]       ;CHECK FOR BIT 21 CLEAR (NO TB MISS)
U 0D59, 08D4,74 ;11848          JMP [LOOP.T15.4]         ;ERROR LOOP IF ENABLED
```

```
                    ;11849
U 0D5A, FF82,15     ;11850          INC LS[ERROR.NUMBER]            ;ERROR 6
U 0D5B, 5F5F,15     ;11851          MCOM LS[BIT15] TO WR[2]         ;CLEAR BIT 15 IN WR
                    ;11852  REPEAT.T15.6:
U 0D5C, 3E13,15     ;11853          MOV WR[2] TO LS[T9]              ;SET UP TAG FIELD IN LS (SHIFTING 0
                    ;11854                                          ; FROM BIT 15 TO BIT 30)
U 0D5D, 3736,95     ;11855          MOV LS[#7FFF8000] TO WR[1]       ;GOT TO CLEAR TB ADDRESS BITS (8-14 AND
                    ;11856                                          ; BIT 31)
U 0D5E, A0C2,95     ;11857          COM WR[1]                       ;NOW BITS 0-14 AND 31 SET IN WR
U 0D5F, EF12,95     ;11858          XOR WR[1] TO LS[T9]              ;CLEAR TB ADDRESS BITS IN LS 9
U 0D60, 3612,15     ;11859          MOV LS[T9] TO WR[0]              ;GET VALUE OF LVA 15-30
U 0D61, BE88,15     ;11860          MOV WR[0] TO LS[ADDRESS.DATA]    ;PRINT VALUE OF LVA 15 THROUGH 30 UNDER
                    ;11861                                          ; OTHER DATA
                    ;11862  LOOP.T15.6:
U 0D62, B66A,95     ;11863          MOV LS[TB.MISS] TO WR[1]         ;EXPECTED DATA (TB MISS SET)
U 0D63, 9813,75     ;11864          MEM.REQ[TEST.V.RCHK] ADRS[T9] DT[LONG]
                    ;11865                                          ;SET UP TO ISSUE TEST.V.RCHK WITH TAG
                    ;11866                                          ; FIELD MISMATCH
U 0D64, 3022,15     ;11867          MOV MEM.DATA TO WR[0]            ;ISSUE MOV TO COMPLETE MEMORY CYCLE
                    ;11868                                          ; (NO NEED TO CHECK THIS DATA)
U 0D65, 9D45,75     ;11869          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;11870                                          ;SET UP TO READ CSR 1
U 0D66, 3022,15     ;11871          MOV MEM.DATA TO WR[0]            ;GET DATA FROM CSR 1
U 0D67, 0869,3C     ;11872          JSR [CHECK.RESULT]              ;CHECK FOR TB MISS SET (BIT 21)
U 0D68, 88D6,24     ;11873          JMP [LOOP.T15.6]                ;ERROR LOOP IF ENABLED
                    ;11874
U 0D69, 23C1,15     ;11875          ROL WR[2]                       ;SHIFT BIT LEFT
                    ;11876          TST WR[2],                      ;SEE IF BIT 31 IS CLEAR
U 0D6A, 2005,35     ;11877           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0D6B, 08D5,C8     ;11878          JMP.IF[N.SET] TO [REPEAT.T15.6] ;REPEAT IF BIT 31 HAS NOT CLEARED YET
                    ;11879                                          ; ELSE GO ON
                    ;11880
U 0D6C, FF82,15     ;11881          INC LS[ERROR.NUMBER]            ;ERROR 7
U 0D6D, 6512,15     ;11882          CLR LS[T9]                      ;CLEAR STARTING TB ADDRESS
U 0D6E, 6588,15     ;11883          CLR LS[ADDRESS.DATA]            ;CLEAR ADDRESS TO PRINT
                    ;11884  REPEAT.T15.7.NEWADDRESS:
U 0D6F, 365F,15     ;11885          MOV LS[BIT15] TO WR[2]           ;USE WR 2 TO STORE SHIFTING BIT PATTERN
                    ;11886  REPEAT.T15.7:
U 0D70, 3612,15     ;11887          MOV LS[T9] TO WR[0]              ;GET TB ADDRESS
U 0D71, AEC4,15     ;11888          BIS WR[2] TO WR[0]               ;SET TAG FIELD IN ADDRESS
U 0D72, BE14,15     ;11889          MOV WR[0] TO LS[T10]             ;DATA FOR TB WRITE STORED IN LS
                    ;11890  LOOP.T15.7:
U 0D73, 9814,F5     ;11891          MEM.REQ[WRITE.TB] ADRS[T10] DT[LONG]
                    ;11892                                          ;SET UP TO WRITE TB WITH SHIFTING 1'S
                    ;11893                                          ; IN TAG FIELD
U 0D74, B272,15     ;11894          WRITE.MEM LS[BIT25]              ;WRITE DATA AT TB ADDRESS SPECIFIED IN
                    ;11895                                          ; LS 9 (BYTE OFFSET SET, OTHERS CLEAR)
U 0D75, 2F82,95     ;11896          CLR WR[1]                       ;EXPECTED DATA (TB MISS CLEAR)
U 0D76, 9815,75     ;11897          MEM.REQ[TEST.V.RCHK] ADRS[T10] DT[LONG]
                    ;11898                                          ;SET UP TO ISSUE TEST.V.RCHK WITH TAG
                    ;11899                                          ; FIELD MATCH
U 0D77, 3022,15     ;11900          MOV MEM.DATA TO WR[0]            ;ISSUE MOV TO COMPLETE MEMORY CYCLE
                    ;11901                                          ; (NO NEED TO CHECK THIS DATA)
U 0D78, 9D45,75     ;11902          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;11903                                          ;SET UP TO READ CSR 1
```

```
U 0D79, 3022,15 :11904          MOV MEM.DATA TO WR[0]              ;GET DATA FROM CSR 1
U 0D7A, 0869,3C :11905          JSR [CHECK.RESULT]                 ;CHECK FOR TB MISS CLEAR (BIT 21)
U 0D7B, 88D7,34 :11906          JMP [LOOP.T15.7]                   ;ERROR LOOP IF ENABLED
                :11907
U 0D7C, 23C1,15 :11908          ROL WR[2]                          ;SHIFT BIT LEFT
                :11909          TST WR[2],                         ;SEE IF BIT 31 IS SET
U 0D7D, 2005,35 :11910           DT(LONG)&SET.ALU.CC               ; AND SET CONDITION CODES
U 0D7E, 08D7,00 :11911          JMP.IF[N.CLR] TO [REPEAT.T15.7]    ;REPEAT IF BIT 31 HAS NOT SET YET
                :11912                                             ; ELSE GO TO NEXT ADDRESS AND REPEAT
U 0D7F, 0A18,2C :11913          JSR [NEXT.TB.ADDRESS]              ;NEXT TB ADDRESS IN LS 9
U 0D80, 08D6,F4 :11914          JMP [REPEAT.T15.7.NEWADDRESS]      ;REPEAT TEST AT NEW ADDRESS IF HERE
                :11915                                             ; IF DONE ALL ADDRESSES, SUBROUTINE
                :11916                                             ; RETURNS TO NEXT INSTRUCTION
                :11917
U 0D81, FF82,15 :11918          INC LS[ERROR.NUMBER]               ;ERROR 8
U 0D82, 6512,15 :11919          CLR LS[T9]                         ;CLEAR STARTING TB ADDRESS
U 0D83, 6588,15 :11920          CLR LS[ADDRESS.DATA]               ;CLEAR ADDRESS TO PRINT IF ERROR
                :11921  REPEAT.T15.8.NEWADDRESS:
U 0D84, 5F5F,15 :11922          MCOM LS[BIT15] TO WR[2]            ;USE WR 2 TO STORE SHIFTING 0 PATTERN
                :11923  REPEAT.T15.8:
U 0D85, 2004,15 :11924          MOV WR[2] TO WR[0]                 ;GET TAG FIELD
U 0D86, 3736,95 :11925          MOV LS[#7FFF8000] TO WR[1]         ;GOT TO CLEAR TB ADDRESS BITS (8-14 AND
                :11926                                             ; BIT 31)
U 0D87, A0C2,95 :11927          COM WR[1]                          ;NOW BITS 0-14 AND BIT 31 ARE SET
U 0D88, AF42,15 :11928          BIC WR[1] TO WR[0]                 ;CLEAR TB ADDRESS BITS
U 0D89, 4712,15 :11929          BIS LS[T9] TO WR[0]                ;SET TB ADDRESS WITHIN TAG FIELD DATA
U 0D8A, BE14,15 :11930          MOV WR[0] TO LS[T10]               ;DATA FOR TB WRITE STORED IN LS
                :11931  LOOP.T15.8:
U 0D8B, 9814,F5 :11932          MEM.REQ[WRITE.TB] ADRS[T10] DT[LONG]
                :11933                                             ;SET UP TO WRITE TB WITH SHIFTING 0'S
                :11934                                             ; IN TAG FIELD
U 0D8C, B272,15 :11935          WRITE.MEM LS[BIT25]                ;WRITE DATA AT TB ADDRESS SPECIFIED IN
                :11936                                             ; LS 9 (BYTE OFFSET SET, OTHERS CLEAR)
U 0D8D, 2F82,95 :11937          CLR WR[1]                          ;EXPECTED DATA (TB MISS CLEAR)
U 0D8E, 9815,75 :11938          MEM.REQ[TEST.V.RCHK] ADRS[T10] DT[LONG]
                :11939                                             ;SET UP TO ISSUE TEST.V.RCHK WITH TAG
                :11940                                             ; FIELD MATCH
U 0D8F, 3022,15 :11941          MOV MEM.DATA TO WR[0]              ;ISSUE MOV TO COMPLETE MEMORY CYCLE
                :11942                                             ; (NO NEED TO CHECK THIS DATA)
U 0D90, 9D45,75 :11943          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                :11944                                             ;SET UP TO READ CSR 1
U 0D91, 3022,15 :11945          MOV MEM.DATA TO WR[0]              ;GET DATA FROM CSR 1
U 0D92, 0869,3C :11946          JSR [CHECK.RESULT]                 ;CHECK FOR TB MISS CLEAR (BIT 21)
U 0D93, 08D8,B4 :11947          JMP [LOOP.T15.8]                   ;ERROR LOOP IF ENABLED
                :11948
U 0D94, 23C1,15 :11949          ROL WR[2]                          ;SHIFT BIT LEFT
                :11950          TST WR[2],                         ;SEE IF BIT 31 IS CLEAR
U 0D95, 2005,35 :11951           DT(LONG)&SET.ALU.CC               ; AND SET CONDITION CODES
U 0D96, 88D8,58 :11952          JMP.IF[N.SET] TO [REPEAT.T15.8]    ;REPEAT IF BIT 31 HAS NOT CLEARED YET
                :11953                                             ; ELSE GO TO NEXT ADDRESS AND REPEAT
U 0D97, 0A18,2C :11954          JSR [NEXT.TB.ADDRESS]              ;NEXT TB ADDRESS IN LS 9
U 0D98, 08D8,44 :11955          JMP [REPEAT.T15.8.NEWADDRESS]      ;REPEAT TEST AT NEW ADDRESS IF HERE
                :11956                                             ; IF DONE ALL ADDRESSES, SUBROUTINE
                :11957                                             ; RETURNS TO NEXT INSTRUCTION
                :11958
```

```
U 0D99, FF82,15 ;11959          INC LS[ERROR.NUMBER]            ;ERROR 9
U 0D9A, 6512,15 ;11960          CLR LS[T9]                      ;CLEAR TB ADDRESS IN LS
                ;11961  BACK.T15.9:
U 0D9B, 9812,F5 ;11962          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                ;11963                                          ;SET UP TO WRITE BACKGROUND IN TB
U 0D9C, B272,15 ;11964          WRITE.MEM LS[BIT25]             ;WRITE BACKGROUND OF 0'S EXCEPT FOR
                ;11965                                          ; BYTE OFFSET BIT (WHICH MUST BE A 1)
U 0D9D, 0A18,2C ;11966          JSR [NEXT.TB.ADDRESS]           ;GET NEXT TB ADDRESS
U 0D9E, 88D9,B4 ;11967          JMP [BACK.T15.9]                ;HERE IF NOT DONE ALL ADDRESSES TO REPEAT
                ;11968                                          ; ELSE SUBROUTINE RETURNS TO NEXT IN-
                ;11969                                          ; STRUCTION
U 0D9F, 6512,15 ;11970          CLR LS[T9]                      ;CLEAR TB ADDRESS
U 0DA0, 6588,15 ;11971          CLR LS[ADDRESS.DATA]            ;CLEAR ADDRESS TO PRINT IF ERROR
U 0DA1, 3651,95 ;11972          MOV LS[BIT8] TO WR[3]           ;SET BIT 8 OF WR 3.  ROL ON WR 3 WILL
                ;11973                                          ; PRODUCE FIRST SHIFTED ADDRESS (RAM
                ;11974                                          ; ADDRESS 1(H))
U 0DA2, B67F,15 ;11975          MOV LS[BIT31] TO WR[2]          ;SET BIT 31 OF WR 2.  ROL ON WR 2 WILL
                ;11976                                          ; PRODUCE FIRST SHIFTED ADDRESS TO
                ;11977                                          ; PRINT (1(H))
                ;11978  REPEAT.T15.9:
U 0DA3, 3612,15 ;11979          MOV LS[T9] TO WR[0]             ;GET TB ADDRESS TO ACCESS
U 0DA4, 3736,95 ;11980          MOV LS[#7FFF8000] TO WR[1]      ;PUT 7FFF8000 IN WR 1
U 0DA5, AEC2,15 ;11981          BIS WR[1] TO WR[0]              ;SET BITS 15-30 OF TB ADDRESS.  LEAVE
                ;11982                                          ; BIT 31 AS IT WAS IN TB ADDRESS
U 0DA6, BE14,15 ;11983          MOV WR[0] TO LS[T10]            ;STORE IN LS FOR TEST.V.RCHK ON COMP-
                ;11984                                          ; LEMENTED DATA
                ;11985  LOOP.T15.9:
U 0DA7, 2F82,95 ;11986          CLR WR[1]                       ;EXPECTED DATA (TB MISS CLEAR INDICATES
                ;11987                                          ; MATCH)
U 0DA8, 9813,75 ;11988          MEM.REQ[TEST.V.RCHK] ADRS[T9] DT[LONG]
                ;11989                                          ;SET UP TO CHECK TAG FIELD FOR MATCH
U 0DA9, 3022,15 ;11990          MOV MEM.DATA TO WR[0]           ;ISSUED TO COMPLETE U-CYCLE (NO NEED TO
                ;11991                                          ; CHECK THIS DATA)
U 0DAA, 9D45,75 ;11992          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;11993                                          ;SET UP TO READ CSR 1
U 0DAB, 3022,15 ;11994          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 0DAC, 0869,3C ;11995          JSR [CHECK.RESULT]              ;CHECK TB MISS FOR A 0 (TAG FIELD MATCH)
U 0DAD, 88DA,74 ;11996          JMP [LOOP.T15.9]                ;ERROR LOOP IF ENABLED
                ;11997
                ;11998  LOOP.T15.9A:
U 0DAE, 2F82,95 ;11999          CLR WR[1]                       ;EXPECTED DATA (TB MISS CLEAR INDICATES
                ;12000                                          ; MATCH)
U 0DAF, 9814,F5 ;12001          MEM.REQ[WRITE.TB] ADRS[T10] DT[LONG]
                ;12002                                          ;SET UP TO WRITE COMPLEMENT DATA
U 0DB0, B272,15 ;12003          WRITE.MEM LS[BIT25]             ;WRITE DATA AT TB ADDRESS SPECIFIED IN
                ;12004                                          ; LS 9 (BYTE OFFSET SET, OTHERS CLEAR)
U 0DB1, 9815,75 ;12005          MEM.REQ[TEST.V.RCHK] ADRS[T10] DT[LONG]
                ;12006                                          ;SET UP TO CHECK TAG FIELD FOR MATCH
U 0DB2, 3022,15 ;12007          MOV MEM.DATA TO WR[0]           ;ISSUED TO COMPLETE U-CYCLE (NO NEED TO
                ;12008                                          ; CHECK THIS DATA)
U 0DB3, 9D45,75 ;12009          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;12010                                          ;SET UP TO READ CSR 1
U 0DB4, 3022,15 ;12011          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 0DB5, 0869,3C ;12012          JSR [CHECK.RESULT]              ;CHECK TB MISS FOR A 0 (TAG FIELD MATCH)
U 0DB6, 88DA,E4 ;12013          JMP [LOOP.T15.9A]               ;ERROR LOOP IF ENABLED
```

```
                    ;12014
U 0DB7, A3C1,95     ;12015          ROL WR[3]                           ;SHIFT ADDRESS TO PUT A 1 IN NEXT BIT
                    ;12016          BIT LS[BIT0] WITH WR[3],            ;SEE IF BIT 0 IS SET (MEANS BIT 31 WAS
                    ;12017                                              ; WAS JUST TESTED)
U 0DB8, 5941,B5     ;12018           DT(LONG)&SET.ALU.CC                ; AND SET CONDITION CODES
U 0DB9, 88DC,11     ;12019          JMP.IF[BIT.SET] TO [TEST.T15.A]     ;GO TO NEXT TEST IF DONE
                    ;12020          BIT LS[BIT15] WITH WR[3],           ;SEE IF DONE BITS 5-0 (IF LAST TEST WAS
                    ;12021                                              ; WITH BIT 14 SET, AFTER ROL BIT 15
                    ;12022                                              ; WILL SET)
U 0DBA, 595F,B5     ;12023           DT(LONG)&SET.ALU.CC                ; SET CONDITION CODES
U 0DBB, 5B00,09     ;12024          SKIP.IF[BITS.CLR]                   ;SKIP NEXT INSTRUCTION IF BIT 15 NOT SET
U 0DBC, 367F,95     ;12025          MOV LS[BIT31] TO WR[3]              ;ELSE SET BIT 31 FOR NEXT ADDRESS BIT
                    ;12026                                              ; AND CLEAR BIT 15
U 0DBD, BE13,95     ;12027          MOV WR[3] TO LS[T9]                 ;STORE ADDRESS
U 0DBE, 23C1,15     ;12028          ROL WR[2]                           ;ADDRESS INFO TO PRINT IF ERROR
U 0DBF, 3E89,15     ;12029          MOV WR[2] TO LS[ADDRESS.DATA]       ;STORE IN LS
U 0DC0, 08DA,34     ;12030          JMP [REPEAT.T15.9]                  ;REPEAT UNTIL ALL ADDRESS BITS HAVE BEEN
                    ;12031                                              ; TESTED
                    ;12032  TEST.T15.A:
U 0DC1, FF82,15     ;12033          INC LS[ERROR.NUMBER]                ;ERROR A
U 0DC2, B640,15     ;12034          MOV LS[#1] TO WR[0]                 ;GET A 1 IN WR 0
U 0DC3, BEFC,15     ;12035          MOV WR[0] TO LS[SIZE]               ;LOAD SIZE REG WITH CODE FOR DATA TYPE
                    ;12036                                              ; OF WORD
U 0DC4, 6512,15     ;12037          CLR LS[T9]                          ;CLEAR TB ADDRESS IN LS
                    ;12038  BACK.T15.A:
U 0DC5, 9812,F5     ;12039          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                    ;12040                                              ;SET UP TO WRITE BACKGROUND IN TB
U 0DC6, B272,15     ;12041          WRITE.MEM LS[BIT25]                 ;WRITE BACKGROUND OF 0'S EXCEPT FOR
                    ;12042                                              ; BYTE OFFSET BIT (WHICH MUST BE A 1)
U 0DC7, 0A18,2C     ;12043          JSR [NEXT.TB.ADDRESS]               ;GET NEXT TB ADDRESS
U 0DC8, 08DC,54     ;12044          JMP [BACK.T15.A]                    ;HERE IF NOT DONE ALL ADDRESSES TO REPEAT
                    ;12045                                              ; ELSE SUBROUTINE RETURNS TO NEXT IN-
                    ;12046                                              ; STRUCTION
                    ;12047
U 0DC9, E580,15     ;12048          CLR LS[CONTROL.STATUS]              ;DISABLE 'OTHER' PRINTOUT
U 0DCA, 3653,15     ;12049          MOV LS[BIT9] TO WR[2]               ;SET UP INCREMENTER (INCREMENTS ADDRESS
                    ;12050                                              ; BIT 0 AT RAM CHIP)
U 0DCB, B653,95     ;12051          MOV LS[BIT9] TO WR[3]               ;BIT WE ARE GOING TO TOGGLE AT MAX RATE
                    ;12052                                              ; FIRST BIT TO TOGGLE IS RAM BIT 0
                    ;12053  REPEAT.T15.A.1:
U 0DCC, 6512,15     ;12054          CLR LS[T9]                          ;START AT ADDRESS 0 IN TB
                    ;12055  REPEAT.T15.A.NEXT:
U 0DCD, 3612,15     ;12056          MOV LS[T9] TO WR[0]                 ;GET TB ADDRESS
U 0DCE, 3736,95     ;12057          MOV LS[#7FFF8000] TO WR[1]          ;GET TAG BITS TO SET IN WR 1
U 0DCF, AEC2,15     ;12058          BIS WR[1] TO WR[0]                  ;ADDRESS WITH TAG FIELD ALL ONES
U 0DD0, BE14,15     ;12059          MOV WR[0] TO LS[T10]                ;STORE IN LS
                    ;12060  LOOP.T15.A.1:
U 0DD1, 2F82,95     ;12061          CLR WR[1]                           ;EXPECTED DATA
U 0DD2, 9813,75     ;12062          MEM.REQ[TEST.V.RCHK] ADRS[T9] DT[LONG]
                    ;12063                                              ;SET UP TO CLOCK CSR1 WITH TB MISS
U 0DD3, 3022,15     ;12064          MOV MEM.DATA TO WR[0]               ;ISSUE MOV TO COMPLETE CYCLE (NO NEED
                    ;12065                                              ; TO CHECK THIS DATA)
U 0DD4, 9D45,75     ;12066          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
U 0DD5, 3022,15     ;12067          MOV MEM.DATA TO WR[0]               ;CHECK FOR TB MISS CLEAR
U 0DD6, 0869,3C     ;12068          JSR [CHECK.RESULT]                  ;CHECK FOR ERRORS
```

```
U ODD7, 08DD,14 ;12069          JMP [LOOP.T15.A.1]                  ;ERROR LOOP IF ENABLED
                ;12070
U ODD8, 9814,F5 ;12071          MEM.REQ[WRITE.TB] ADRS[T10] DT[LONG]
                ;12072                                              ;SET UP TO WRITE COMPLEMENT IN TAG
U ODD9, B272,15 ;12073          WRITE.MEM LS[BIT25]                 ;WRITE A ONE IN BYTE OFFSET
                ;12074
                ;12075          ADD WR[3] TO LS[T9],                ;TOGGLE BIT IN ADDRESS
U ODDA, EC13,D5 ;12076           DT(SIZE)&SET.ALU.CC                ; AND SET CONDITION CODES
U ODDB, 08DC,D0 ;12077          JMP.IF[N.CLR] TO [REPEAT.T15.A.NEXT] ;IF BIT 15 DIDN'T SET, REPEAT AT
                ;12078                                              ; NEXT ADDRESS
U ODDC, EC13,15 ;12079          ADD WR[2] TO LS[T9]                 ;ELSE INCREMENT ADDRESS BELOW TOGGLED
                ;12080                                              ; BIT
U ODDD, B65E,15 ;12081          MOV LS[BIT15] TO WR[0]              ;GOING TO CLEAR BIT 15 IN LS 9
U ODDE, 6F12,15 ;12082          XOR WR[0] TO LS[T9]                 ;CLEAR BIT 15 IN ADDRESS
                ;12083          BIT WR[3] WITH LS[T9],              ;IF TOGGLED BIT IS NOW SET, WE'RE DONE
U ODDF, D913,B5 ;12084           DT(LONG)&SET.ALU.CC                ; SET CONDITION CODES
U ODE0, 08DC,D9 ;12085          JMP.IF[BITS.CLR] TO [REPEAT.T15.A.NEXT] ;REPEAT TEST AT NEW ADDRESS IF NOT
                ;12086                                              ; DONE
                ;12087
U ODE1, 3612,15 ;12088          MOV LS[T9] TO WR[0]                 ;GET ADDRESS FROM LS
U ODE2, C528,15 ;12089          BIC LS[#FFFF] TO WR[0]              ;CLEAR BITS 9 THROUGH 14 (ALONG WITH
                ;12090                                              ; OTHERS THAT ARE ALREADY CLEAR)
                ;12091          XOR LS[BIT31] TO WR[0],             ;TOGGLE BIT 31
U ODE3, 437E,35 ;12092           DT(LONG)&SET.ALU.CC                ; AND SET CONDITION CODES
U ODE4, BE12,15 ;12093          MOV WR[0] TO LS[T9]                 ;PUT NEW ADDRESS IN LS 9
U ODE5, 88DC,D8 ;12094          JMP.IF[N.SET] TO [REPEAT.T15.A.NEXT] ;IF BIT 31 WAS CLEAR, REPEAT USING
                ;12095                                              ; SAME TOGGLED BIT BUT WITH BIT 31 SET
                ;12096                                              ; ELSE GOING TO START DESCENDING ADDRS
                ;12097
U ODE6, 6512,15 ;12098                  CLR LS[T9]                  ;START AT ADDRESS 0 IN TB (WILL BE
                ;12099                                              ; COMPLEMENTED TO START AT TOP ADDRESS)
                ;12100  REPEAT.T15.A.NEXT1:
U ODE7, 7D12,15 ;12101                  COM LS[T9]                  ;COMPLEMENT ADDRESS TO GET DESCENDING
                ;12102                                              ; ADDRESSES
U ODE8, 3612,15 ;12103                  MOV LS[T9] TO WR[0]         ;GET CURRENT ADDRESS
U ODE9, 3736,95 ;12104                  MOV LS[#7FFF8000] TO WR[1]  ;GET TAG BITS TO CLEAR IN WR 1
U ODEA, AF42,15 ;12105                  BIC WR[1] TO WR[0]          ;CLEAR TAG BITS WITHIN TB ADDRESS
U ODEB, BE14,15 ;12106                  MOV WR[0] TO LS[T10]        ;AND STORE IN LS
                ;12107
                ;12108  LOOP.T15.A.2:
U ODEC, 2F82,95 ;12109                  CLR WR[1]                   ;EXPECTED DATA
U ODED, 9813,75 ;12110                  MEM.REQ[TEST.V.RCHK] ADRS[T9] DT[LONG]
                ;12111                                              ;SET UP CLOCK CSR WITH TB MISS
U ODEE, 3022,15 ;12112                  MOV MEM.DATA TO WR[0]       ;ISSUE MOV TO COMPLETE U-CYCLE (NO NEED
                ;12113                                              ; TO CHECK THIS DATA)
U ODEF, 9D45,75 ;12114                  MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;12115                                              ;SET UP TO READ CSR
U ODF0, 3022,15 ;12116                  MOV MEM.DATA TO WR[0]       ;GET RESULT FROM CSR 1
U ODF1, 0869,3C ;12117                  JSR [CHECK.RESULT]          ;CHECK FOR TB MISS CLEAR
U ODF2, 88DE,C4 ;12118                  JMP [LOOP.T15.A.2]          ;ERROR LOOP IF ENABLED
                ;12119
U ODF3, 9814,F5 ;12120                  MEM.REQ[WRITE.TB] ADRS[T10] DT[LONG]
                ;12121                                              ;SET UP TO WRITE COMPLEMENT IN TAG
U ODF4, B272,15 ;12122                  WRITE.MEM LS[BIT25]         ;WRITE A 1 IN BYTE OFFSET
                ;12123
```

```
U 0DF5, 7D12,15 ;12124          COM LS[T9]                      ;RECOMPLEMENT ADDRESS TO GET ORIGINAL
                ;12125          ADD WR[3] TO LS[T9],            ;TOGGLE BIT IN ADDRESS
U 0DF6, EC13,D5 ;12126           DT(SIZE)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0DF7, 88DE,70 ;12127          JMP.IF[N.CLR] TO [REPEAT.T15.A.NEXT1] ;IF BIT 15 DIDN'T SET, REPEAT AT
                ;12128                                          ; NEXT ADDRESS
U 0DF8, EC13,15 ;12129          ADD WR[2] TO LS[T9]             ;ELSE INCREMENT ADDRESS BELOW TOGGLED
                ;12130                                          ; BIT
U 0DF9, B65E,15 ;12131          MOV LS[BIT15] TO WR[0]          ;GOING TO CLEAR BIT 15 IN LS 9
U 0DFA, 6F12,15 ;12132          XOR WR[0] TO LS[T9]             ;CLEAR BIT 15 IN ADDRESS
                ;12133          BIT WR[3] WITH LS[T9],          ;IF TOGGLED BIT IS NOW SET, WE'RE DONE
U 0DFB, D913,B5 ;12134           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 0DFC, 88DE,79 ;12135          JMP.IF[BITS.CLR] TO [REPEAT.T15.A.NEXT1] ;REPEAT TEST AT NEW ADDRESS IF
                ;12136                                          ; NOT DONE
                ;12137
U 0DFD, 3612,15 ;12138          MOV LS[T9] TO WR[0]             ;GET ADDRESS FROM LS
U 0DFE, C528,15 ;12139          BIC LS[#FFFF] TO WR[0]          ;CLEAR BITS 9 THROUGH 14 (ALONG WITH
                ;12140                                          ; OTHERS THAT ARE ALREADY CLEAR)
                ;12141          XOR LS[BIT31] TO WR[0],         ;TOGGLE BIT 31
U 0DFF, 437E,35 ;12142           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0E00, BE12,15 ;12143          MOV WR[0] TO LS[T9]             ;PUT NEW ADDRESS IN LS 9
U 0E01, 08DE,78 ;12144          JMP.IF[N.SET] TO [REPEAT.T15.A.NEXT1] ;IF BIT 31 WAS CLEAR, REPEAT USING
                ;12145                                          ; SAME TOGGLED BIT BUT WITH BIT 31 SET
                ;12146
U 0E02, A3C1,95 ;12147          ROL WR[3]                       ;NEXT BIT TO TOGGLE AT MAX RATE
                ;12148          BIT LS[BIT15] WITH WR[3],       ;SEE IF BIT 15 HAS SET
U 0E03, 595F,B5 ;12149           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0E04, 88DC,C9 ;12150          JMP.IF[BITS.CLR] TO [REPEAT.T15.A.1] ;REPEAT TEST WITH NEW TOGGLED BIT
                ;12151                                          ; IF BIT 15 HAS NOT SET
                ;12152                                          ; ELSE GOING TO TOGGLE BIT 31 NEXT
                ;12153
U 0E05, 367F,95 ;12154          MOV LS[BIT31] TO WR[3]          ;BIT WE ARE GOING TO TOGGLE AT MAX RATE
                ;12155                                          ; BIT TO TOGGLE NOW IS RAM BIT 6
                ;12156  REPEAT.T15.A1.1:
U 0E06, 6512,15 ;12157          CLR LS[T9]                      ;START AT ADDRESS 0 IN TB
                ;12158  REPEAT.T15.A1.NEXT:
U 0E07, 3612,15 ;12159          MOV LS[T9] TO WR[0]             ;GET TB ADDRESS
U 0E08, 3736,95 ;12160          MOV LS[#7FFF8000] TO WR[1]      ;GET TAG BITS TO SET IN WR 1
U 0E09, AEC2,15 ;12161          BIS WR[1] TO WR[0]              ;ADDRESS WITH TAG FIELD ALL ONES
U 0E0A, BE14,15 ;12162          MOV WR[0] TO LS[T10]            ;STORE IN LS
                ;12163  LOOP.T15.A1.1:
U 0E0B, 2F82,95 ;12164          CLR WR[1]                       ;EXPECTED DATA
U 0E0C, 9813,75 ;12165          MEM.REQ[TEST.V.RCHK] ADRS[T9] DT[LONG]
                ;12166                                          ;SET UP TO CLOCK CSR1 WITH TB MISS
U 0E0D, 3022,15 ;12167          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE CYCLE (NO NEED
                ;12168                                          ; TO CHECK THIS DATA)
U 0E0E, 9D45,75 ;12169          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
U 0E0F, 3022,15 ;12170          MOV MEM.DATA TO WR[0]           ;CHECK FOR TB MISS CLEAR
U 0E10, 0869,3C ;12171          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 0E11, 88E0,B4 ;12172          JMP [LOOP.T15.A1.1]             ;ERROR LOOP IF ENABLED
                ;12173
U 0E12, 9814,F5 ;12174          MEM.REQ[WRITE.TB] ADRS[T10] DT[LONG]
                ;12175                                          ;SET UP TO WRITE COMPLEMENT IN TAG
U 0E13, B272,15 ;12176          WRITE.MEM LS[BIT25]             ;WRITE A ONE IN BYTE OFFSET
                ;12177
                ;12178          XOR WR[3] TO LS[T9],            ;TOGGLE BIT IN ADDRESS
```

```
U 0E14, EF13,B5 ;12179          DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 0E15, 88E0,78 ;12180          JMP.IF[N.SET] TO [REPEAT.T15.A1.NEXT] ;IF BIT 31 DIDN'T CLEAR, REPEAT
                ;12181                                          ; AT NEXT ADDRESS
                ;12182          ADD WR[2] TO LS[T9],            ;ELSE INCREMENT ADDRESS BELOW TOGGLED
                ;12183                                          ; BIT
U 0E16, 6C13,55 ;12184          DT(SIZE)&SET.ALU.CC             ; AND SET CONDITION CODES
U 0E17, 08E0,70 ;12185          JMP.IF[N.CLR] TO [REPEAT.T15.A1.NEXT] ;REPEAT TEST AT NEW ADDRESS IF NOT
                ;12186                                          ; DONE
                ;12187
U 0E18, 6512,15 ;12188          CLR LS[T9]                      ;START AT ADDRESS 0 IN TB (WILL BE
                ;12189                                          ; COMPLEMENTED TO FORM TOP ADDRESS)
                ;12190  REPEAT.T15.A1.NEXT1:
U 0E19, 7D12,15 ;12191          COM LS[T9]                      ;COMPLEMENT ADDRESS TO FORM DESCENDING
                ;12192                                          ; ADDRESSES
U 0E1A, 3612,15 ;12193          MOV LS[T9] TO WR[0]             ;GET CURRENT ADDRESS
U 0E1B, 3736,95 ;12194          MOV LS[#7FFF8000] TO WR[1]      ;GET TAG BITS TO CLEAR IN WR 1
U 0E1C, AF42,15 ;12195          BIC WR[1] TO WR[0]              ;CLEAR TAG BITS WITHIN TB ADDRESS
U 0E1D, BE14,15 ;12196          MOV WR[0] TO LS[T10]            ;AND STORE IN LS
                ;12197
                ;12198  LOOP.T15.A1.2:
U 0E1E, 2F82,95 ;12199          CLR WR[1]                       ;EXPECTED DATA
U 0E1F, 9813,75 ;12200          MEM.REQ[TEST.V.RCHK] ADRS[T9] DT[LONG]
                ;12201                                          ;SET UP CLOCK CSR WITH TB MISS
U 0E20, 3022,15 ;12202          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE (NO NEED
                ;12203                                          ; TO CHECK THIS DATA)
U 0E21, 9D45,75 ;12204          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;12205                                          ;SET UP TO READ CSR
U 0E22, 3022,15 ;12206          MOV MEM.DATA TO WR[0]           ;GET RESULT FROM CSR 1
U 0E23, 0869,3C ;12207          JSR [CHECK.RESULT]              ;CHECK FOR TB MISS CLEAR
U 0E24, 08E1,E4 ;12208          JMP [LOOP.T15.A1.2]             ;ERROR LOOP IF ENABLED
                ;12209
U 0E25, 9814,F5 ;12210          MEM.REQ[WRITE.TB] ADRS[T10] DT[LONG]
                ;12211                                          ;SET UP TO WRITE COMPLEMENT IN TAG
U 0E26, B272,15 ;12212          WRITE.MEM LS[BIT25]             ;WRITE A 1 IN BYTE OFFSET
                ;12213
U 0E27, 7D12,15 ;12214          COM LS[T9]                      ;RECOMPLEMENT ADDRESS TO GET ORIGINAL
                ;12215          XOR WR[3] TO LS[T9],            ;TOGGLE BIT IN ADDRESS
U 0E28, EF13,B5 ;12216           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0E29, 88E1,98 ;12217          JMP.IF[N.SET] TO [REPEAT.T15.A1.NEXT1] ;IF BIT 31 DIDN'T CLEAR, REPEAT
                ;12218                                          ; AT NEXT ADDRESS
                ;12219          ADD WR[2] TO LS[T9],            ;ELSE INCREMENT ADDRESS BELOW TOGGLED
                ;12220                                          ; BIT
U 0E2A, 6C13,55 ;12221           DT(SIZE)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0E2B, 08E1,90 ;12222          JMP.IF[N.CLR] TO [REPEAT.T15.A1.NEXT1];REPEAT TEST AT NEW ADDRESS IF NOT DONE
                ;12223                                          ; ELSE, DONE!!
                ;12224
U 0E2C, FF82,15 ;12225          INC LS[ERROR.NUMBER]            ;ERROR B
U 0E2D, B670,15 ;12226          MOV LS[OTHER.DATA] TO WR[0]     ;SET BIT IN WR 0 TO PRINT 'OTHER' DATA
U 0E2E, 3E80,15 ;12227          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT IN CONTROL WORD
U 0E2F, 6588,15 ;12228          CLR LS[ADDRESS.DATA]            ;CLEAR ADDRESS TO PRINT IF ERROR
U 0E30, 989C,F5 ;12229          MEM.REQ[WRITE.TB] ADRS[ZERO] DT[LONG]
                ;12230                                          ;SET UP TO WRITE TB ADDRESS 0 WITH 0
                ;12231                                          ; IN TAG FIELD
U 0E31, B272,15 ;12232          WRITE.MEM LS[BIT25]             ;WRITE TB WITH 1 IN BYTE OFFSET
                ;12233  LOOP.T15.B:
```

```
U 0E32, 9B63,F5 ;12234          MEM.REQ[WRITE.UBS.MAP] ADRS[BIT17] DT[LONG]
                ;12235                                 ;SET UP TO WRITE TO UBS MAP.  TAG AT
                ;12236                                 ; TAG RAM ADDRESS 0 WILL HAVE TAG 2 SET
                ;12237                                 ; IF DISABLE ON UBS MAP WRITE FAILS
U 0E33, B272,15 ;12238          WRITE.MEM LS[BIT25]    ;SET BYTE OFFSET IN TB
U 0E34, 2F82,95 ;12239          CLR WR[1]              ;EXPECTED DATA (TB MISS CLEAR)
U 0E35, 989D,75 ;12240          MEM.REQ[TEST.V.RCHK] ADRS[ZERO] DT[LONG]
                ;12241                                 ;CLOCK CSR WITH RESULTANT TB MISS
U 0E36, 3022,15 ;12242          MOV MEM.DATA TO WR[0]  ;ISSUE TO COMPLETE U-CYCLE (NO NEED TO
                ;12243                                 ; CHECK THIS DATA]
U 0E37, 9D45,75 ;12244          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;12245                                 ;READ CSR1
U 0E38, 3022,15 ;12246          MOV MEM.DATA TO WR[0]  ;GET RESULT
U 0E39, 0869,3C ;12247          JSR [CHECK.RESULT]     ;CHECK FOR TB MISS CLEAR
U 0E3A, 88E3,24 ;12248          JMP [LOOP.T15.B]       ;ERROR LOOP IF ENABLED
                ;12249
U 0E3B, FF82,15 ;12250          INC LS[ERROR.NUMBER]   ;ERROR C
U 0E3C, 989C,F5 ;12251          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;12252                                 ;SET UP TO WRITE TB ADDRESS 0 WITH 0'S
                ;12253                                 ; IN TAG FIELD
U 0E3D, B29C,15 ;12254          WRITE.MEM LS[#0]       ;WRITE TB WITH 0'S IN ALL BITS, INCLUD-
                ;12255                                 ; ING BYTE OFFSET
                ;12256
                ;12257  LOOP.T15.C:
U 0E3E, 989D,75 ;12258          MEM.REQ[TEST.V.RCHK] ADRS[#0] DT[LONG]
                ;12259                                 ;SET UP TO CLOCK CSR 1 WITH TB MISS BIT
U 0E3F, 3022,15 ;12260          MOV MEM.DATA TO WR[0]  ;ISSUE MOV TO COMPLETE U-CYCLE (NO NEED
                ;12261                                 ; TO CHECK THIS DATA)
U 0E40, B66A,95 ;12262          MOV LS[TB.MISS] TO WR[1] ;EXPECTED DATA (TB MISS SET SINCE BYTE
                ;12263                                 ; OFFSET IS CLEAR)
U 0E41, 9D45,75 ;12264          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;12265                                 ;SET UP TO READ CSR 1
U 0E42, 3022,15 ;12266          MOV MEM.DATA TO WR[0]  ;GET DATA FROM CSR 1
U 0E43, 0869,3C ;12267          JSR [CHECK.RESULT]     ;CHECK FOR BIT 21 SET (TB MISS ASSERTED
U 0E44, 88E3,E4 ;12268          JMP [LOOP.T15.C]       ;ERROR LOOP IF ENABLED
                ;12269  END.T15:
```

```
;12270  .PAGE   " TB Parity And CSR 1 Valid Bit Tests"
;12271
;12272  .toc    "  TEST 16 TB Parity Test Including TB P0 Bit of TB RAM (M8391  MCT Module)"
;12273  ;++
;12274  ;
;12275  ; Test Description:
;12276  ;
;12277  ;       This test checks the TB parity logic and the only untested TB RAM bit;
;12278  ;       TB P0 L.  The TB P0 L bit in the RAM can not be read directly as the
;12279  ;       other bits can, so it must be tested by setting or clearing it at each
;12280  ;       location and checking that a parity error does not occur on a read of
;12281  ;       the TB.  The parity logic must be tested first.  TB P0 L is produced
;12282  ;       by the UB CSR 2 PAL on a write to the TB. The inputs used to produce TB
;12283  ;       P0 L are VALID H, GEN P0 H and PROT PAR H.  VALID H comes directly from
;12284  ;       the TB RAM.  GEN P0 H comes from a pair of parity generators with the
;12285  ;       PA bits from RAM, BYTE OFFSET from RAM and TB PAR DIAG H (diagnostic
;12286  ;       bit to force a parity error) as its inputs.  PROT PAR H comes from a
;12287  ;       PROM with PROT A through PROT D and MODIFY H as its inputs.  TB PAR ERR
;12288  ;       is generated by the UB CSR 2 PAL from GEN P0, PROT PAR, VALID, and TB
;12289  ;       P0 and sent to the CSR 1A PAL.  TEST.V.RCHK will be used to clock the
;12290  ;       CSR so this bit can be checked.  The TEST.V.RCHK data path is described
;12291  ;       in the TB miss test preceeding.
;12292  ;         TB PAR ERR L will be checked first by writing the TB with all 0's and
;12293  ;       issueing a MEM.REQ with MF=TEST.V.RCHK.  Then CSR 1 is read and bit 15
;12294  ;       (TB PAR ERR H) is checked for a low.  Then the TEST.V.RCHK is repeated
;12295  ;       with CSR 1 bit 29 (TB PAR DIAG H) set and TB PAR ERR H is checked for a
;12296  ;       high.  This is repeated with all 1's and then a shifting 1 pattern
;12297  ;       across all the TB RAM bits.
;12298  ;         The TB P0 RAM bit will then be checked at all locations by writing a
;12299  ;       1 or a 0 and checking that no parity error occurs.  A simple march test
;12300  ;       will complete the test.
;12301  ;
;12302  ;
;12303  ; Assumptions:
;12304  ;
;12305  ;       It is assumed that all previous tests have run successfully.
;12306  ;
;12307  ; Test Steps:
;12308  ;
;12309  ;       1)  Set up error number and module number in LS (for error printouts)
;12310  ;           and clear previous error number in LS.  Also set MME in CSR 1 and
;12311  ;           clear other bits.
;12312  ;       2)  Set up error mask to check bit 15 only (CSR TB PAR ERR).
;12313  ;       3)  Execute MEM.REQ with MF=WRITE.TB and DT=longword.  Use an LS loc-
;12314  ;           tion containing all zeros as the address.  Issue WRITE.MEM LS with
;12315  ;           data of all 0's.
;12316  ;       4)  Clear WR 1.
;12317  ;       5)  Execute MEM.REQ with MF=TEST.V.RCHK and DT=longword.  Use an LS
;12318  ;           location containing all 0's as the address.  Issue MOV MEM.DATA TO
;12319  ;           WR[0] to complete the TEST.V.RCHK function (no need to check the
;12320  ;           data coming back).
;12321  ;       6)  Issue MEM.REQ with MF=READ.CSR and MOV MEM.DATA TO WR[0] to read CSR
;12322  ;           1.  Check bit 15 for a 0 (no parity error).
;12323  ;       7)  Write CSR 1 with a 1 in bit 29 (TB PAR DIAG H) to force bad parity.
;12324  ;       8)  Repeat steps 5 and 6 checking bit 15 for a 1 (parity error).
```

```
;12325  ;       9)  Repeat steps 3 through 8 with TB data of all 1's in step 3.
;12326  ;       10) Repeat steps 3 through 8 with TB data of shifting 1's in step 3 un-
;12327  ;           til all TB outputs have had a 1 shifted across.
;12328  ;       11) Repeat steps 3 through 8 with TB data of 1 in PA bit 09 and 1 in
;12329  ;           PROT A.
;12330  ;       12) Clear the TB parity diagnostic bit in CSR 1.  Load a temporary LS
;12331  ;           location with 0's to be used as the TB RAM address.  Clear WR 1
;12332  ;           (expected data).
;12333  ;       13) Execute MEM.REQ with MF=WRITE.TB and DT=longword.  Use the tempor-
;12334  ;           ary LS loction as the address.  Issue WRITE.MEM LS with data of all
;12335  ;           0's.
;12336  ;       14) Execute MEM.REQ with MF=TEST.V.RCHK and DT=longword.  Use the temp-
;12337  ;           orary LS location as the address.  Issue MOV MEM.DATA TO WR[0] to
;12338  ;           complete the TEST.V.RCHK function (no need to check the data coming
;12339  ;           back).
;12340  ;       15) Issue MEM.REQ with MF=READ.CSR and MOV MEM.DATA TO WR[0] to read CSR
;12341  ;           1.  Check bit 15 for a 0 (no parity error).
;12342  ;       16) Repeat steps 13 through 15, incrementing the temporary LS location
;12343  ;           to access all addresses.
;12344  ;       17) Repeat steps 13 through 16 with data of a 1 in PA 09 as the data
;12345  ;           pattern to write in the RAM.  This forces TB P0 L to a 1.
;12346  ;       18) Run a simple march pattern using steps 13 through 16 above.  Do a
;12347  ;           read/write/read pattern, with the read checking for no parity er-
;12348  ;           ror.
;12349  ;
;12350  ; Errors:
;12351  ;
;12352  ;       NOTE:  Expected and received data is TB PAR ERR bit in CSR 1.
;12353  ;
;12354  ;       Error 1 - TB PAR ERR was asserted when no parity error occured. TB data
;12355  ;                 of all 0's.
;12356  ;       Error 2 - TB PAR ERR failed to be asserted on a TB parity error.  TB
;12357  ;                 data of all 0's.
;12358  ;       Error 3 - TB PAR ERR was asserted when no parity error occured. TB data
;12359  ;                 of all 1's.
;12360  ;       Error 4 - TB PAR ERR failed to be asserted on a TB parity error.  TB
;12361  ;                 data of all 1's.
;12362  ;
;12363  ;       NOTE:  In errors 5 through A, the data written to the TB is displayed
;12364  ;              under OTHER in the error report.  Bits 0-14 go to PA bits 09-23
;12365  ;              respectively.  Bits 25-30 go to byte offset, modify, prot A,
;12366  ;              prot B, prot C, and prot D bits respectively.
;12367  ;
;12368  ;       Error 5 - TB PAR ERR was asserted when no parity error occured. TB data
;12369  ;                 of shifting 1's on PA and BYTE OFFSET bits.  See note above.
;12370  ;       Error 6 - TB PAR ERR failed to be asserted on a TB parity error.  TB
;12371  ;                 data of shifting 1's on PA and BYTE OFFSET bits.  See note
;12372  ;                 above.
;12373  ;       Error 7 - TB PAR ERR was asserted when no parity error occured. TB data
;12374  ;                 of shifting 1's on PROT and MODIFY bits.  See note above.
;12375  ;       Error 8 - TB PAR ERR failed to be asserted on a TB parity error.  TB
;12376  ;                 data of shifting 1's on PROT and MODIFY bits.  See note above.
;12377  ;       Error 9 - TB PAR ERR was asserted when no parity error occured. TB data
;12378  ;                 of 1 in one PA bit and 1 in one PROT bit.  See note above.
;12379  ;       Error A - TB PAR ERR failed to be asserted on a TB parity error.  TB
```

```
;12380  ;                   data of 1 in one PA bit and 1 in one PROT bit.  See note above.
;12381  ;       Error B - TB RAM failed when trying to write a 0 in TB P0 L.  Ram ad-
;12382  ;                 dress is under 'OTHER DATA'.
;12383  ;       Error C - TB RAM failed when trying to write a 1 in TB P0 L.  RAM ad-
;12384  ;                 dress is under 'OTHER DATA'.
;12385  ;       Error D - TB RAM failed march test.  RAM address is under 'OTHER DATA'.
;12386  ;
;12387  ; Debug:
;12388  ;
;12389  ;       Two methods of debug will be explained.  If the MCT module is in a
;12390  ;       test station then the MCT can be single stepped making debug easier
;12391  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;12392  ;       method will be explained also.  When single stepping the memory con-
;12393  ;       troller will help in debug, this section will explicitly suggest it.
;12394  ;
;12395  ;       Error 1 - This error indicates a problem with the UB CSR 2 PAL, CSR 1
;12396  ;       PAL or other parity logic. The best place to start is at the CSR 2 PAL.
;12397  ;       Single step the CPU to the MOV MEM.DATA TO WR[0] that follows the TEST.V
;12398  ;       .RCHK memory request.  The memory u-code will be looping waiting for
;12399  ;       CPU GRANT to be dropped.  Examine the output of UB CSR 2 for a high on
;12400  ;       TB PAR ERR L.  If high, check it at the input to the CSR 1A PAL.  If
;12401  ;       high at the CSR 1A PAL then CSR 1A is probably bad.
;12402  ;         If TB PAR ERR L is low, check the inputs to the UB CSR 2 PAL for
;12403  ;       a low on GEN P0 H, PROT PAR H, TB P0 L, and VALID H.  If these are OK,
;12404  ;       then the UB CSR 2 PAL is probably bad.
;12405  ;         If GEN P0 H is high, check it at the parity generator.  If high there
;12406  ;       then check the inputs for all lows on the inputs.  If the input from
;12407  ;       the other parity generator is high, check its inputs for all lows.  If
;12408  ;       the inputs are OK, suspect the parity generator itself.
;1240C  ;         If PROT PAR H is high, check it at the output of the PROM that pro-
;12410  ;       duces it.  If high there check the inputs for lows on PROT A through
;12411  ;       PROT D and MODIFY H.  If the inputs are low, the PROM is probably bad.
;12412  ;         If TB P0 L is high, check it at the output of the RAM.  A failure
;12413  ;       there could mean a bad RAM or a bad UB CSR 2 PAL which produces it on
;12414  ;       the write.  If MCT single step is available, step the CPU to the MEM
;12415  ;       .REQ with MF=WRITE.TB, enable MCT single step, and step the CPU to the
;12416  ;       WRITE.MEM LS instruction.  Step the MCT to the cycle that asserts TB
;12417  ;       WE L and check for a low on TB P0 L at the input of the RAM.  If this
;12418  ;       is OK, then the RAM is bad.  If not, check that TB WE L is low at the
;12419  ;       UB CSR 2 PAL along with lows on VALID H, GEN P0 H and PROT PAR H.  If
;12420  ;       these are OK, the output TB P0 L should be low.  Otherwise the PAL is
;12421  ;       bad.
;12422  ;
;12423  ;       Error 2 - This error indicates a problem with the UB CSR 2 PAL, the
;12424  ;       GEN P0 H parity generators, or the input TB PAR DIAG H.  Single step
;12425  ;       the CPU to the MOV MEM.DATA TO WR[0] that follows the TEST.V.RCHK memory
;12426  ;       request.  The memory u-code will be looping waiting for CPU GRANT to be
;12427  ;       dropped.  Examine the output of UB CSR 2 for a low on TB PAR ERR L.  If
;12428  ;       low, check it at the input to the CSR 1A PAL.  If high at the CSR 1A
;12429  ;       PAL then CSR 1A is probably bad.
;12430  ;         If TB PAR ERR L is high, check the input to the UB CSR 2 PAL for a
;12431  ;       high on GEN P0 H.  The other inputs are the same as error 1 above so
;12432  ;       are probably OK.  If GEN P0 L is high, the UB CSR 2 PAL is most sus-
;12433  ;       pect.
;12434  ;         If GEN P0 L is low, check TB PAR DIAG H going to the parity generator
```

```
                    ;12435  ;       for a high.  If this is OK, the output of this generator should be high
                    ;12436  ;       and the output of the second parity generator, GEN PO H should be high.
                    ;12437  ;       Otherwise the parity generator is probably bad.
                    ;12438  ;
                    ;12439  ;       Error 3 - This is similar to error 1, except all inputs to the parity
                    ;12440  ;       generators (except TB PAR DIAG H) and to the PROT PAR PROM are high.
                    ;12441  ;       The other signals which are different are PROT PAR H which should be
                    ;12442  ;       high, and VALID H which should be high.
                    ;12443  ;
                    ;12444  ;       Error 4 - The most likely suspect is the UB CSR 2 PAL.
                    ;12445  ;
                    ;12446  ;       Error 5 - This error is similar to error 1, except one of the PA inputs
                    ;12447  ;       or the BYTE OFFSET input to the parity generators should be high.  The
                    ;12448  ;       signals that are different are GEN PO H which should be high and TB PO
                    ;12449  ;       L which should be high.
                    ;12450  ;
                    ;12451  ;       Error 6 - The most likely suspect is the UB CSR 2 PAL.
                    ;12452  ;
                    ;12453  ;       Error 7 - The most likely suspect is the PROT PAR PROM.
                    ;12454  ;
                    ;12455  ;       Errors 8 through A - The most likely suspect is the UB CSR 2 PAL.
                    ;12456  ;
                    ;12457  ;       Errors B through D - The most likely suspect is the TB RAM itself.
                    ;12458  ;       The failing address is printed under 'OTHER DATA'.
                    ;12459  ;
                    ;12460  ;--
                    ;12461
                    ;12462  T.16:
                    ;12463
U 0E45, B65E,15     ;12464          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;12465                                          ; STATUS WORD
U 0E46, 3E80,15     ;12466          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;12467                                          ; BIT 15 INDICATES START OF TEST TO
                    ;12468                                          ; CONSOLE
U 0E47, 10E0,15     ;12469          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;12470  WAIT.T16.0:
U 0E48, 08E4,84     ;12471          JMP [WAIT.T16.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;12472                                          ; RESPOND
                    ;12473
U 0E49, 0A1A,AC     ;12474          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;12475
U 0E4A, DF5E,15     ;12476          MCOM LS[TB.PAR.ERR] TO WR[0]    ;CLEAR BIT 15, SET OTHERS IN WR
U 0E4B, 3E8A,15     ;12477          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK BIT 15 (TB PARITY ERROR) ONLY
U 0E4C, B640,15     ;12478          MOV LS[#1] TO WR[0]             ;1 IN WR
U 0E4D, BEFC,15     ;12479          MOV WR[0] TO LS[SIZE]           ;SET UP SIZE REG FOR DATA TYPE OF WORD
U 0E4E, 6512,15     ;12480          CLR LS[T9]                      ;ADDRESS IN TB TO WRITE DATA IN
                    ;12481  T16.1:
U 0E4F, 6514,15     ;12482          CLR LS[T10]                     ;DATA OF ALL 0'S TO WRITE IN TB
U 0E50, 08EA,BC     ;12483          JSR [T16.NO.PAR.ERR]            ;GO TEST FOR NO PARITY ERROR
                    ;12484  T16.2:
U 0E51, FF82,15     ;12485          INC LS[ERROR.NUMBER]            ;ERROR 2
U 0E52, 08EB,6C     ;12486          JSR [T16.PAR.ERR]               ;GO TEST FOR PARITY ERROR
                    ;12487  T16.3:
U 0E53, FF82,15     ;12488          INC LS[ERROR.NUMBER]            ;ERROR 3
U 0E54, 7D14,15     ;12489          COM LS[T10]                     ;DATA PATTERN OF ALL 1'S TO WRITE IN TB
```

```
U 0E55, 08EA,BC ;12490          JSR [T16.NO.PAR.ERR]            ;GO TEST FOR NO PARITY ERROR
                ;12491  T16.4:
U 0E56, FF82,15 ;12492          INC LS[ERROR.NUMBER]            ;ERROR 4
U 0E57, 08EB,6C ;12493          JSR [T16.PAR.ERR]               ;GO TEST FOR PARITY ERROR
                ;12494
                ;12495
U 0E58, FF82,15 ;12496          INC LS[ERROR.NUMBER]            ;ERROR 5
U 0E59, B670,15 ;12497          MOV LS[OTHER.DATA] TO WR[0]     ;SET BIT TO TELL CONSOLE TO PRINT 'OTHER''
U 0E5A, 3E80,15 ;12498          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT IN CONTROL AND STATUS WORD
U 0E5B, 3641,15 ;12499          MOV LS[#1] TO WR[2]             ;PUT A 1 IN WR 2 (GOES TO PA 09)
                ;12500  REPEAT.T16.5:
U 0E5C, 3E89,15 ;12501          MOV WR[2] TO LS[ADDRESS.DATA]   ;PUT DATA UNDER OTHER IN ERROR REPORT
U 0E5D, BE21,15 ;12502          MOV WR[2] TO LS[10]             ;LOAD TEMP LS LOCATION WITH SHIFTING
                ;12503                                          ; 1'S PATTERN (STARTS AT LVA 0 WHICH
                ;12504                                          ; GOES TO PA 9)
U 0E5E, 08EA,BC ;12505          JSR [T16.NO.PAR.ERR]            ;GO TEST FOR NO PARITY ERROR
                ;12506
U 0E5F, 23C1,15 ;12507          ROL WR[2]                       ;NEXT PATTERN
                ;12508          BIT LS[BIT26] WITH WR[2],       ;SEE IF BIT 26 IS SET (MEANS WE HAVE
                ;12509                                          ; DONE BYTE OFFSET (BIT 25))
U 0E60, 5975,35 ;12510           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0E61, 08E6,61 ;12511          JMP.IF[BIT.SET] TO [T16.6]      ;GO TO NEXT TEST IF BYTE OFFSET IS DONE
                ;12512                                          ; ELSE SEE IF PA BITS DONE
                ;12513          BIT LS[BIT15] WITH WR[2],       ;SEE IF BIT 15 HAS SET (MEANS DONE PA
                ;12514                                          ; 09 THROUGH 23 IF SET)
U 0E62, D95F,35 ;12515           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0E63, 5B00,09 ;12516          SKIP.IF[BITS.CLR]               ;SKIP NEXT INSTRUCTION IF BIT 15 NOT SET
U 0E64, B673,15 ;12517          MOV LS[BIT25] TO WR[2]          ;ELSE SET BYTE OFFSET BIT
U 0E65, 08E5,C4 ;12518          JMP [REPEAT.T16.5]              ;REPEAT WITH NEW PATTERN
                ;12519
                ;12520  T16.6:
U 0E66, 3641,15 ;12521          MOV LS[#1] TO WR[2]             ;PUT A 1 IN WR 2 (GOES TO PA 09)
U 0E67, FF82,15 ;12522          INC LS[ERROR.NUMBER]            ;ERROR 6
                ;12523  REPEAT.T16.6:
U 0E68, 3E89,15 ;12524          MOV WR[2] TO LS[ADDRESS.DATA]   ;PUT DATA UNDER OTHER IN ERROR REPORT
U 0E69, BE21,15 ;12525          MOV WR[2] TO LS[10]             ;LOAD TEMP LS LOCATION WITH SHIFTING
                ;12526                                          ; 1'S PATTERN (STARTS AT LVA 0 WHICH
                ;12527                                          ; GOES TO PA 9)
U 0E6A, 9814,F5 ;12528          MEM.REQ[WRITE.TB] ADRS[T10] DT[LONG]
                ;12529                                          ;SET UP TO WRITE TB WITH DATA PATTERN
U 0E6B, B214,15 ;12530          WRITE.MEM LS[T10]               ;WRITE DATA PATTERN IN TB
U 0E6C, 08EB,6C ;12531          JSR [T16.PAR.ERR]               ;GO TO TEST FOR PARITY ERROR
                ;12532
U 0E6D, 23C1,15 ;12533          ROL WR[2]                       ;NEXT PATTERN
                ;12534          BIT LS[BIT26] WITH WR[2],       ;SEE IF BIT 26 IS SET (MEANS WE HAVE
                ;12535                                          ; DONE BYTE OFFSET (BIT 25))
U 0E6E, 5975,35 ;12536           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0E6F, 08E7,41 ;12537          JMP.IF[BIT.SET] TO [T16.7]      ;GO TO NEXT TEST IF BYTE OFFSET IS DONE
                ;12538                                          ; ELSE SEE IF PA BITS DONE
                ;12539          BIT LS[BIT15] WITH WR[2],       ;SEE IF BIT 15 HAS SET (MEANS DONE PA
                ;12540                                          ; 09 THROUGH 23 IF SET)
U 0E70, D95F,35 ;12541           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0E71, 5B00,09 ;12542          SKIP.IF[BITS.CLR]               ;SKIP NEXT INSTRUCTION IF BIT 15 NOT SET
U 0E72, 3671,15 ;12543          MOV LS[BIT24] TO WR[2]          ;ELSE SET BYTE OFFSET BIT
U 0E73, 88E6,84 ;12544          JMP [REPEAT.T16.6]              ;REPEAT WITH NEW PATTERN
```

```
                ;12545
                ;12546  T16.7:
U 0E74, FF82,15 ;12547          INC LS[ERROR.NUMBER]            ;ERROR 7
U 0E75, B675,15 ;12548          MOV LS[BIT26] TO WR[2]          ;SET MODIFY BIT
                ;12549  REPEAT.T16.7:
U 0E76, 3E89,15 ;12550          MOV WR[2] TO LS[ADDRESS.DATA]   ;PUT DATA UNDER OTHER IN ERROR REPORT
U 0E77, BE21,15 ;12551          MOV WR[2] TO LS[10]             ;LOAD TEMP LS LOCATION WITH SHIFTING
                ;12552                                          ; 1'S PATTERN (STARTS AT LVA 26 WHICH
                ;12553                                          ; GOES TO MOD BIT)
U 0E78, 08EA,BC ;12554          JSR [T16.NO.PAR.ERR]            ;GO TEST FOR NO PARITY ERROR
                ;12555
U 0E79, 23C1,15 ;12556          ROL WR[2]                       ;NEXT PATTERN
                ;12557          BIT LS[BIT31] WITH WR[2],       ;SEE IF BIT 31 IS SET (MEANS WE HAVE
                ;12558                                          ; DONE PROT D)
U 0E7A, 597F,35 ;12559           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0E7B, 08E7,69 ;12560          JMP.IF[BITS.CLR] TO [REPEAT.T16.7] ;REPEAT IF PROT D NOT YET SET
                ;12561                                          ; ELSE GO TO NEXT TEST
                ;12562
U 0E7C, FF82,15 ;12563          INC LS[ERROR.NUMBER]            ;ERROR 8
U 0E7D, B675,15 ;12564          MOV LS[BIT26] TO WR[2]          ;SET MODIFY BIT
                ;12565  REPEAT.T16.8:
U 0E7E, 3E89,15 ;12566          MOV WR[2] TO LS[ADDRESS.DATA]   ;PUT DATA UNDER OTHER IN ERROR REPORT
U 0E7F, BE21,15 ;12567          MOV WR[2] TO LS[10]             ;LOAD TEMP LS LOCATION WITH SHIFTING
                ;12568                                          ; 1'S PATTERN (STARTS AT LVA 26 WHICH
                ;12569                                          ; GOES TO MOD BIT)
U 0E80, 9814,F5 ;12570          MEM.REQ[WRITE.TB] ADRS[T10] DT[LONG]
                ;12571                                          ;SET UP TO WRITE TB WITH DATA PATTERN
U 0E81, B214,15 ;12572          WRITE.MEM LS[T10]               ;WRITE DATA PATTERN IN TB
U 0E82, 08EB,6C ;12573          JSR [T16.PAR.ERR]               ;GO TEST FOR PARITY ERROR
                ;12574
U 0E83, 23C1,15 ;12575          ROL WR[2]                       ;NEXT PATTERN
                ;12576          BIT LS[BIT31] WITH WR[2],       ;SEE IF BIT 31 IS SET (MEANS WE HAVE
                ;12577                                          ; DONE PROT D)
U 0E84, 597F,35 ;12578           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0E85, 88E7,E9 ;12579          JMP.IF[BITS.CLR] TO [REPEAT.T16.8] ;REPEAT IF PROT D NOT YET SET
                ;12580                                          ; ELSE GO TO NEXT TEST
                ;12581  T16.9:
U 0E86, FF82,15 ;12582          INC LS[ERROR.NUMBER]            ;ERROR 9
U 0E87, B643,15 ;12583          MOV LS[BIT1] TO WR[2]           ;SET PA 09
U 0E88, 4777,15 ;12584          BIS LS[BIT27] TO WR[2]          ;AND PROT A
U 0E89, 3E15,15 ;12585          MOV WR[2] TO LS[T10]            ;AND STORE IN LS
U 0E8A, 3E89,15 ;12586          MOV WR[2] TO LS[ADDRESS.DATA]   ;PUT DATA UNDER OTHER IN ERROR REPORT
U 0E8B, 08EA,BC ;12587          JSR [T16.NO.PAR.ERR]            ;GO TEST FOR NO PARITY ERROR
                ;12588  T16.A:
U 0E8C, FF82,15 ;12589          INC LS[ERROR.NUMBER]            ;ERROR A
U 0E8D, 08EB,6C ;12590          JSR [T16.PAR.ERR]               ;GO TEST FOR PARITY ERROR
                ;12591
U 0E8E, FF82,15 ;12592          INC LS[ERROR.NUMBER]            ;ERROR B
U 0E8F, 6588,15 ;12593          CLR LS[ADDRESS.DATA]            ;CLEAR ADDRESS TO PRINT IF ERROR
U 0E90, 6514,15 ;12594          CLR LS[T10]                     ;DATA PATTERN IS ALL 0'S
                ;12595  REPEAT.T16.B:
U 0E91, 08EA,BC ;12596          JSR [T16.NO.PAR.ERR]            ;GO TEST FOR NO PARITY ERROR
U 0E92, 0A18,2C ;12597          JSR [NEXT.TB.ADDRESS]           ;GET NEXT TB ADDRESS
U 0E93, 88E9,14 ;12598          JMP [REPEAT.T16.B]              ;REPEAT UNTIL ALL ADDRESS IN TB ACCESSED
                ;12599
```

```
U 0E94, FF82,15 ;12600          INC LS[ERROR.NUMBER]              ;ERROR C
U 0E95, 6512,15 ;12601          CLR LS[T9]                       ;CLEAR ADDRESS IN TB TO ACCESS
U 0E96, 6588,15 ;12602          CLR LS[ADDRESS.DATA]             ;CLEAR ADDRESS TO PRINT IF ERROR
U 0E97, 3642,15 ;12603          MOV LS[BIT1] TO WR[0]            ;SET LVA 00 TO SET PA 09
U 0E98, BE14,15 ;12604          MOV WR[0] TO LS[T10]             ;DATA PATTERN IS ONE BIT SET IN TB
                ;12605                                           ; TO COMPLEMENT PARITY BIT IN TB
                ;12606  REPEAT.T16.C:
U 0E99, 08EA,BC ;12607          JSR [T16.NO.PAR.ERR]             ;GO TEST FOR NO PARITY ERROR
U 0E9A, 0A18,2C ;12608          JSR [NEXT.TB.ADDRESS]            ;GET NEXT TB ADDRESS
U 0E9B, 08E9,94 ;12609          JMP [REPEAT.T16.C]               ;REPEAT UNTIL ALL ADDRESS IN TB ACCESSED
                ;12610
U 0E9C, FF82,15 ;12611          INC LS[ERROR.NUMBER]             ;ERROR D
U 0E9D, 6512,15 ;12612          CLR LS[T9]                       ;CLEAR ADDRESS
                ;12613  BACK.T16.D:
U 0E9E, 9812,F5 ;12614          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                ;12615                                           ;SET UP TO WRITE TB
U 0E9F, B29C,15 ;12616          WRITE.MEM LS[ZERO]               ;WRITE ZEROS IN TB
U 0EA0, 0A18,2C ;12617          JSR [NEXT.TB.ADDRESS]            ;GET NEXT ADDRESS
U 0EA1, 88E9,E4 ;12618          JMP [BACK.T16.D]                 ;REPEAT UNTIL ALL ADDRESS WRITTEN
                ;12619
U 0EA2, 6512,15 ;12620          CLR LS[T9]                       ;CLEAR STARTING ADDRESS
U 0EA3, 6588,15 ;12621          CLR LS[ADDRESS.DATA]             ;CLEAR ADDRESS TO PRINT IF ERROR
U 0EA4, 3642,15 ;12622          MOV LS[BIT1] TO WR[0]            ;SET LVA 00 IN WR
U 0EA5, BE14,15 ;12623          MOV WR[0] TO LS[T10]             ;1 BIT SET IN TB TO TOGGLE PARITY
                ;12624  REPEAT.T16.D:
U 0EA6, 08EA,EC ;12625          JSR [LOOP.T16.NO.PAR.ERR]        ;CHECK FOR NO PARITY ERR AT CURRENT
                ;12626                                           ; ADDRESS
U 0EA7, 08EA,BC ;12627          JSR [T16.NO.PAR.ERR]             ;WRITE TB TO TOGGLE PARITY AND CHECK
                ;12628                                           ; FOR NO PARITY ERROR
U 0EA8, 0A18,2C ;12629          JSR [NEXT.TB.ADDRESS]            ;GET NEW TB ADDRESS
U 0EA9, 08EA,64 ;12630          JMP [REPEAT.T16.D]               ;REPEAT UNTIL ALL TB ADDRESSES CHECKED
                ;12631
U 0EAA, 88EC,14 ;12632          JMP [END.T16]                    ;DONE
                ;12633
                ;12634
                ;12635  T16.NO.PAR.ERR:
U 0EAB, 0A17,DC ;12636          JSR [WRITE.CSR1.MME]             ;ENABLE MEMORY MANAGEMENT
U 0EAC, 9812,F5 ;12637          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                ;12638                                           ;SET UP TO WRITE TB AT ADDRESS IN LS 9
U 0EAD, B214,15 ;12639          WRITE.MEM LS[T10]                ;WRITE DATA PATTERN IN TB
                ;12640  LOOP.T16.NO.PAR.ERR:
U 0EAE, 2F82,95 ;12641          CLR WR[1]                        ;EXPECTED DATA (TB PARITY ERROR CLEAR)
U 0EAF, 9813,75 ;12642          MEM.REQ[TEST.V.RCHK] ADRS[T9] DT[LONG]
                ;12643                                           ;EXECUTE TEST.V.RCHK TO CLOCK CSR
U 0EB0, 3022,15 ;12644          MOV MEM.DATA TO WR[0]            ;ISSUE MOV TO COMPLETE U-CYCLE (NO NEED
                ;12645                                           ; TO CHECK THIS DATA)
U 0EB1, 9D45,75 ;12646          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;12647                                           ;SET UP TO READ CSR 1
U 0EB2, 3022,15 ;12648          MOV MEM.DATA TO WR[0]            ;GET CONTENTS OF CSR 1
U 0EB3, 0869,3C ;12649          JSR [CHECK.RESULT]               ;CHECK TB PARITY ERROR FOR A 0
U 0EB4, 88EA,E4 ;12650          JMP [LOOP.T16.NO.PAR.ERR]        ;ERROR LOOP IF ENABLED
U 0EB5, 5B00,14 ;12651          RETURN                           ;RETURN TO MAIN PROGRAM FLOW
                ;12652
                ;12653
                ;12654  T16.PAR.ERR:
```

```
U 0EB6, B676,15 ;12655          MOV LS[MME] TO WR[0]              ;SET MEMORY MANAGMENT ENABLE BIT IN WR
U 0EB7, C77A,15 ;12656          BIS LS[TB.PAR.DIAG] TO WR[0]      ;SET TB PARITY DIAGNOSTIC BIT
U 0EB8, 8A17,FC ;12657          JSR [WRITE.CSR1]                  ;SET MME AND TB PAR DIAG IN CSR 1
                ;12658  LOOP.T16.PAR.ERR:
U 0EB9, 369E,95 ;12659          MOV LS[ONES] TO WR[1]             ;EXPECTED DATA (TB PARITY ERROR SET)
U 0EBA, 9813,75 ;12660          MEM.REQ[TEST.V.RCHK] ADRS[T9] DT[LONG]
                ;12661                                            ;EXECUTE TEST.V.RCHK TO CLOCK CSR
U 0EBB, 3022,15 ;12662          MOV MEM.DATA TO WR[0]             ;ISSUE MOV TO COMPLETE U-CYCLE (NO NEED
                ;12663                                            ; TO CHECK THIS DATA)
U 0EBC, 9D45,75 ;12664          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;12665                                            ;SET UP TO READ CSR 1
U 0EBD, 3022,15 ;12666          MOV MEM.DATA TO WR[0]             ;GET CONTENTS OF CSR 1
U 0EBE, 0869,3C ;12667          JSR [CHECK.RESULT]                ;CHECK TB PARITY ERROR FOR A 1
U 0EBF, 88EB,94 ;12668          JMP [LOOP.T16.PAR.ERR]            ;ERROR LOOP IF ENABLED
U 0EC0, 5B00,14 ;12669          RETURN                            ;RETURN TO MAIN PROGRAM
                ;12670
                ;12671  END.T16:
```

```
;12672  .PAGE   "  TEST 17 CSR 1 VALID Bit Test (M8391  MCT Module)"
;12673  ;++
;12674  ;
;12675  ; Test Description:
;12676  ;
;12677  ;       This is a quick check of the VALID bit (bit 14) of CSR 1.  The VALID
;12678  ;       bit itself was tested previously by the TB RAM test but the CSR bit
;12679  ;       was not.  The test uses the same micro-flows as the TB PARITY test
;12680  ;       preceeding.  It writes the TB with a value to either set or clear the
;12681  ;       VALID bit in the TB.  Then a TEST.V.RCHK is performed to clock the CSR.
;12682  ;       Finally CSR 1 is read and the VALID bit is checked.  Only bit 14 will
;12683  ;       be checked in this test; the other bits will be masked out.  Note:  The
;12684  ;       VALID bit that is output by the CSR is the complement of the valid bit
;12685  ;       in the TB.  I.E. BUS MC D14 (VALID H) is really VALID ERR H.
;12686  ;
;12687  ; Assumptions:
;12688  ;
;12689  ;       It is assumed that all previous tests have run successfully.
;12690  ;
;12691  ; Test Steps:
;12692  ;
;12693  ;       1)  Set up error number and module number in LS (for error printouts)
;12694  ;           and clear previous error number in LS.  Also set MME in CSR 1.
;12695  ;       2)  Set up error mask to check bit 14 only (VALID H).
;12696  ;       3)  Execute MEM.REQ with MF=WRITE.TB and DT=longword.  Use an LS loc-
;12697  ;           tion containing all zeros as the address.  Issue WRITE.MEM LS with
;12698  ;           data of all 1's.
;12699  ;       4)  Execute MEM.REQ with MF=TEST.V.RCHK and DT=longword.  Use an LS
;12700  ;           location containing all 0's as the address.  Issue MOV MEM.DATA TO
;12701  ;           WR[0] to complete the TEST.V.RCHK function (no need to check the
;12702  ;           data coming back).
;12703  ;       5)  Issue MEM.REQ with MF=READ.CSR and MOV MEM.DATA TO WR[0] to read CSR
;12704  ;           1.  Check bit 14 for a 0 (complement of VALID H in TB).
;12705  ;       6)  Repeat steps 3 through 5 with all 0's as the data to write in the
;12706  ;           TB and check CSR 1 bit 14 for a 1 (VALID bit in TB clear).
;12707  ;
;12708  ; Errors:
;12709  ;
;12710  ;       Error 1 - CSR 1 VALID ERR bit failed to clear when VALID was set in
;12711  ;                 the TB.
;12712  ;       Error 2 - CSR 1 VALID ERR bit failed to set when VALID was clear in
;12713  ;                 the TB.
;12714  ;
;12715  ; Debug:
;12716  ;
;12717  ;       Two methods of debug will be explained.  If the MCT module is in a
;12718  ;       test station then the MCT can be single stepped making debug easier
;12719  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;12720  ;       method will be explained also.  When single stepping the memory con-
;12721  ;       troller will help in debug, this section will explicitly suggest it.
;12722  ;
;12723  ;       Error 1 - This error indicates a problem with the CSR 1B PAL or one of
;12724  ;       its inputs.  Single step the CPU to the MOV MEM.DATA to WR[0] instruc-
;12725  ;       tion after the TEST.V.RCHK.  The microcode will be looping waiting for
;12726  ;       CPU DATA RCVD.  Check the inputs to the CSR 1B PAL for a high on VALID
```

```
                    ;12727  ;       H and ADDR PH L.  If these are OK, check that the input CSR ERR SUM CLK
                    ;12728  ;       H is not floating.  If all inputs are OK, the output BUS MC D14 H (VAL-
                    ;12729  ;       ID) should be low.  If not, the PAL is probably bad.
                    ;12730  ;
                    ;12731  ;       Error 2 - Same as error 1 except VALID H should be low and BUS MC D14 H
                    ;12732  ;       should be high.
                    ;12733  ;
                    ;12734  ;--
                    ;12735
                    ;12736  T.17:
                    ;12737
U 0EC1, B65E,15     ;12738          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;12739                                          ; STATUS WORD
U 0EC2, 3E80,15     ;12740          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;12741                                          ; BIT 15 INDICATES START OF TEST TO
                    ;12742                                          ; CONSOLE
U 0EC3, 10E0,15     ;12743          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;12744  WAIT.T17.0:
U 0EC4, 88EC,44     ;12745          JMP [WAIT.T17.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;12746                                          ; RESPOND
                    ;12747
U 0EC5, 0A1A,AC     ;12748          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;12749
U 0EC6, 5F5C,15     ;12750          MCOM LS[VALID.ERR] TO WR[0]     ;CLEAR BIT 14 IN WR, SET OTHERS
U 0EC7, 3E8A,15     ;12751          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK VALID ERR BIT (BIT 14) ONLY
U 0EC8, 0A17,DC     ;12752          JSR [WRITE.CSR1.MME]            ;ENABLE MEMORY MANAGEMENT
                    ;12753
U 0EC9, 989C,F5     ;12754          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                    ;12755                                          ;SET UP TO WRITE TB
U 0ECA, 329E,15     ;12756          WRITE.MEM LS[ONES]              ;WRITE ONES IN TB ADDRESS 0
                    ;12757  LOOP.T17.1:
U 0ECB, 2F82,95     ;12758          CLR WR[1]                       ;EXPECTED DATA (VALID ERROR CLEAR)
U 0ECC, 989D,75     ;12759          MEM.REQ[TEST.V.RCHK] ADRS[#0] DT[LONG]
                    ;12760                                          ;SET UP TO CLOCK CSR WITH ERROR BITS
U 0ECD, 3022,15     ;12761          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE
U 0ECE, 9D45,75     ;12762          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;12763                                          ;SET UP TO READ CSR 1
U 0ECF, 3022,15     ;12764          MOV MEM.DATA TO WR[C]           ;GET CONTENTS OF CSR 1
U 0ED0, 0869,3C     ;12765          JSR [CHECK.RESULT]              ;CHECK VALID ERROR FOR A 0
U 0ED1, 88EC,B4     ;12766          JMP [LOOP.T17.1]                ;ERROR LOOP IF ENABLED
                    ;12767
U 0ED2, FF82,15     ;12768          INC LS[ERROR.NUMBER]            ;ERROR 2
U 0ED3, 989C,F5     ;12769          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                    ;12770                                          ;SET UP TO WRITE TB
U 0ED4, B29C,15     ;12771          WRITE.MEM LS[ZERO]              ;WRITE ZEROS IN TB ADDRESS 0
                    ;12772  LOOP.T17.2:
U 0ED5, 369E,95     ;12773          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA (VALID ERROR SET)
U 0ED6, 989D,75     ;12774          MEM.REQ[TEST.V.RCHK] ADRS[#0] DT[LONG]
                    ;12775                                          ;SET UP TO CLOCK CSR WITH ERROR BITS
U 0ED7, 3022,15     ;12776          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE
U 0ED8, 9D45,75     ;12777          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;12778                                          ;SET UP TO READ CSR 1
U 0ED9, 3022,15     ;12779          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 0EDA, 0869,3C     ;12780          JSR [CHECK.RESULT]              ;CHECK VALID ERROR FOR A 1
U 0EDB, 88ED,54     ;12781          JMP [LOOP.T17.2]                ;ERROR LOOP IF ENABLED
```

```
;12782  END.T17·
```

```
;12783  .PAGE   " Protection Prom Tests"
;12784
;12785  .toc    "  TEST 18 TEST.V.WCHK Test (for use in next test) (M8391  MCT Module)"
;12786  ;++
;12787  ;
;12788  ; Test Description:
;12789  ;
;12790  ;       This is a short test designed to test the micro-path of the TEST.V.WCHK
;12791  ;       memory instruction.  The instruction is similar (and uses part of the
;12792  ;       same u-flow) to the TEST.V.RCHK function used previously.  The only
;12793  ;       difference is that it asserts RD CSR (C47) before clocking the CSR so
;12794  ;       that WR ACROSS PG ERR will be asserted on a PAGE BOUNDARY.  The WR
;12795  ;       ACROSS PG ERR bit will be tested later.  This test just checks that the
;12796  ;       CSR is clocked.
;12797  ;         The test will load the TB to set the BYTE OFFSET bit and issue a MEM
;12798  ;       .REQ with MF=TEST.V.WCHK.  A MOV MEM.DATA TO WR is issued, but this data
;12799  ;       is discarded.  The CSR is read and TB PAR ERR is checked for 0.  The
;12800  ;       CSR TB PAR DIAG bit is now set and the test repeated.  TB PAR ERR is
;12801  ;       checked for a 1.  This verifies that the TEST.V.WCHK is clocking the
;12802  ;       CSR correctly and not going off into never-never land, thus verifing
;12803  ;       the u-flow.
;12804  ;         The TEST.V.WCHK u-code is as follows:  The MEM.REQ dispatches as des-
;12805  ;       cribed in the beginning of this listing.  At the dispatch address, ROT
;12806  ;       CLK is issued, and VAR BYPASS and memory busy are enabled.  The next
;12807  ;       cycle keeps VAR bypass and memory busy asserted, and asserts RD CSR.
;12808  ;       The following cycle repeats this and also clocks the CSR.  The next
;12809  ;       cycle keeps memory busy asserted and enables the PA bits onto the MC
;12810  ;       BUS to be read by the CPU.  The next cycle drops memory busy and keeps
;12811  ;       the data enabled on the MC bus until CPU GRANT is dropped.  It then
;12812  ;       prepares for a UNIBUS cycle and returns to the idle loop.
;12813  ;
;12814  ; Assumptions:
;12815  ;
;12816  ;       It is assumed that all previous tests have run successfully.
;12817  ;
;12818  ; Test Steps:
;12819  ;
;12820  ;       1)  Set up error number and module number in LS (for error printouts)
;12821  ;           and clear previous error number in LS.  Also set MME bit in CSR 1
;12822  ;           and clear other bits.
;12823  ;       2)  Set up error mask to check bit 15 only (CSR TB PAR ERR).
;12824  ;       3)  Execute MEM.REQ with MF=WRITE.TB and DT=longword.  Use an LS loc-
;12825  ;           tion containing all zeros as the address.  Issue WRITE.MEM LS with
;12826  ;           data of all 0's.
;12827  ;       4)  Clear WR 1.
;12828  ;       5)  Execute MEM.REQ with MF=TEST.V.WCHK and DT=longword.  Use an LS
;12829  ;           location containing all 0's as the address.  Issue MOV MEM.DATA TO
;12830  ;           WR[0] to complete the TEST.V.RCHK function (no need to check the
;12831  ;           data coming back).
;12832  ;       6)  Issue MEM.REQ with MF=READ.CSR.  Issue MOV MEM.DATA TO WR[0] to read
;12833  ;           CSR 1.  Check bit 15 for a 0 (no parity error).
;12834  ;       7)  Write CSR 1 with a 1 in bit 29 (TB PAR DIAG H) to force bad parity.
;12835  ;       8)  Repeat steps 5 and 6 checking bit 15 for a 1 (parity error).
;12836  ;
;12837  ; Errors:
```

```
                ;12838  ;
                ;12839  ;       Error 1 - TEST.V.WCHK failed to clock CSR 1 to clear TB parity.
                ;12840  ;       Error 2 - TEST.V.WCHK failed to clock CSR 1 to set TB parity.
                ;12841  ;
                ;12842  ; Debug:
                ;12843  ;
                ;12844  ;       Two methods of debug will be explained.  If the MCT module is in a
                ;12845  ;       test station then the MCT can be single stepped making debug easier
                ;12846  ;       in some cases.  Otherwise only the CPU can be single stepped and this
                ;12847  ;       method will be explained also.  When single stepping the memory con-
                ;12848  ;       troller will help in debug, this section will explicitly suggest it.
                ;12849  ;
                ;12850  ;       Error 1 and 2 - This error indicates a micro-flow problem since all hardware
                ;12851  ;       used by this test has been previously tested.  If MCT single step is
                ;12852  ;       available, the micro-flow can be traced.  Otherwise a logic analyzer
                ;12853  ;       is required.
                ;12854  ;
                ;12855  ;--
                ;12856
                ;12857  T.18:
                ;12858
U OEDC, B65E,15 ;12859          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;12860                                          ; STATUS WORD
U OEDD, 3E80,15 ;12861          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;12862                                          ; BIT 15 INDICATES START OF TEST TO
                ;12863                                          ; CONSOLE
U OEDE, 10E0,15 ;12864          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;12865  WAIT.T18.0:
U OEDF, 88ED,F4 ;12866          JMP [WAIT.T18.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;12867                                          ; RESPOND
                ;12868
U OEE0, 0A1A,AC ;12869          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                ;12870
U OEE1, DF5E,15 ;12871          MCOM LS[TB.PAR.ERR] TO WR[0]    ;CLEAR BIT 15, SET OTHERS IN WR
U OEE2, 3E8A,15 ;12872          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK BIT 15 (TB PARITY ERROR) ONLY
U OEE3, 0A17,DC ;12873          JSR [WRITE.CSR1.MME]            ;ENABLE MEMORY MANAGEMENT
                ;12874
U OEE4, 989C,F5 ;12875          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;12876                                          ;SET UP TO WRITE TB AT ADDRESS 0
U OEE5, B29C,15 ;12877          WRITE.MEM LS[ZERO]              ;WRITE DATA PATTERN IN TB
                ;12878  LOOP.T18.1:
U OEE6, 2F82,95 ;12879          CLR WR[1]                       ;EXPECTED DATA (TB PARITY ERROR CLEAR)
U OEE7, 199C,F5 ;12880          MEM.REQ[TEST.V.WCHK] ADRS[#0] DT[LONG]
                ;12881                                          ;EXECUTE TEST.V.WCHK TO CLOCK CSR
U OEE8, 3022,15 ;12882          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE (NO NEED
                ;12883                                          ; TO CHECK THIS DATA)
U OEE9, 9D45,75 ;12884          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;12885                                          ;SET UP TO READ CSR 1
U OEEA, 3022,15 ;12886          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U OEEB, 0869,3C ;12887          JSR [CHECK.RESULT]              ;CHECK TB PARITY ERROR FOR A 0
U OEEC, 88EE,64 ;12888          JMP [LOOP.T18.1]                ;ERROR LOOP IF ENABLED
                ;12889
U OEED, FF82,15 ;12890          INC LS[ERROR.NUMBER]            ;ERROR 2
U OEEE, B676,15 ;12891          MOV LS[MME] TO WR[0]            ;SET MEMORY MANAGMENT ENABLE BIT IN WR
U OEEF, C77A,15 ;12892          BIS LS[TB.PAR.DIAG] TO WR[0]    ;SET TB PARITY DIAGNOSTIC BIT
```

```
U 0EF0, 8A17,FC ;12893          JSR [WRITE.CSR1]                ;SET MME AND TB PAR DIAG IN CSR 1
                ;12894  LOOP.T18.2:                                                                  U
U 0EF1, 369E,95 ;12895          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA (TB PARITY ERROR SET)  U
U 0EF2, 199C,F5 ;12896          MEM.REQ[TEST.V.WCHK] ADRS[#0] DT[LONG]
                ;12897                                          ;EXECUTE TEST.V.WCHK TO CLOCK CSR     U
U 0EF3, 3022,15 ;12898          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE (NO NEED
                ;12899                                          ; TO CHECK THIS DATA)
U 0EF4, 9D45,75 ;12900          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;12901                                          ;SET UP TO READ CSR 1
U 0EF5, 3022,15 ;12902          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 0EF6, 0869,3C ;12903          JSR [CHECK.RESULT]              ;CHECK TB PARITY ERROR FOR A 1
U 0EF7, 88EF,14 ;12904          JMP [LOOP.T18.2]                ;ERROR LOOP IF ENABLED
                ;12905  END.T18:
```

```
;12906  .PAGE   "  TEST 19 Protection Prom CSR 1 Access/Modify Refused Test (M8391  MCT Module)"
;12907  ;++
;12908  ;
;12909  ; Test Description:
;12910  ;
;12911  ;       This test checks the Protection PROM and the associated CSR 1 error
;12912  ;       bits ACCESS REFUSED and MODIFY REFUSED.  The test will check every ad-
;12913  ;       dress of the protection PROM that can be accessed by using all proces-
;12914  ;       sor modes and three of the four possible memory function types.  One
;12915  ;       memory function type does not use the protection prom outputs (the CSR
;12916  ;       is not clocked with the information).  These memory function types are
;12917  ;       those with MF0 and MF1 both high.
;12918  ;         The protection prom uses the four PROT bits and the MODIFY bit as
;12919  ;       five of its address bits.  Two more address bits come from the CPU as
;12920  ;       the current mode (kernel, executive, supervisor, or user).  The remain-
;12921  ;       ing two address bits come from the lower two bits of the memory func-
;12922  ;       tion (MF) field sent over and latched during the MEM.REQ instruction.
;12923  ;       The outputs of the prom, ACCESS REFUSED L and MODIFY REFUSED L, go to
;12924  ;       the CSR 1A PAL and are clocked into bits 22 and 23 respectively by the
;12925  ;       memory u-code.  The CSR will be read to check these bits.
;12926  ;         The first part of the prom tested is the NO CHECK portion.  The CPU
;12927  ;       first writes the PROT and MODIFY bits in the TB and sets up the current
;12928  ;       mode in the PSL.  The TB PAR DIAG bit is set in CSR 1 to force an error
;12929  ;       sum when the next MEM.REQ is executed.  The CPU sends a MEM.REQ with MF
;12930  ;       =WRITE.V.NOCHK.  The memory dispatches as described in the beginning of
;12931  ;       this listing.  The data path is as follows:  At the dispatch address
;12932  ;       VAR bypass and memory busy are enabled, data from the CPU is enabled
;12933  ;       onto the MC bus, DATI is set, and RAS is enabled.  The next cycle keeps
;12934  ;       memory busy asserted and trys to read the array.  The following cycle
;12935  ;       clocks CSR 1, and sets up other signals to read the array.  The next
;12936  ;       cycle continues to read the array and branches on ERR SUM set. The fol-
;12937  ;       lowing cycles keep memory busy set and finally branches on CPU DATA REQ
;12938  ;       (sent by a WRITE.MEM LS instruction from the CPU).  The next cycle goes
;12939  ;       to the UB preparatory cycle, then to the idle loop.  Now the CSR is
;12940  ;       read.  The ACCESS REFUSED and MODIFY REFUSED bits are checked for 0.
;12941  ;       This is repeated for each protection code,  modify bit, and current
;12942  ;       mode setting.  MF0=0 and MF1=0 for a WRITE.V.NOCHK.
;12943  ;         The next two tests (read access check and write access check) will
;12944  ;       use constants loaded in LS to determine the result.  First the PROT,
;12945  ;       MODIFY, and current mode bits are set up by the CPU.  Then a MEM.REQ
;12946  ;       with MF=TEST.V.RCHK is executed to load the CSR with the ACCESS and
;12947  ;       MODIFY REFUSED bits.  The CSR is checked for the correct value.  This
;12948  ;       is repeated for each protection code and modify bit setting and in each
;12949  ;       current mode setting.  MF0=0 and MF1=1 for a TEST.V.RCHK.
;12950  ;         The last test is similar to the previous paragraph except a MEM.REQ
;12951  ;       with MF=TEST.V.WCHK is issued.  MF0=1 and MF1=0 for a TEST.V.WCHK.
;12952  ;
;12953  ; Assumptions:
;12954  ;
;12955  ;       It is assumed that all previous tests have run successfully.
;12956  ;
;12957  ; Test Steps:
;12958  ;
;12959  ;       1)  Set up error number and module number in LS (for error printouts)
;12960  ;           and clear previous error number in LS.
```

```
;12961  ;       2)  Set up error mask to check bits 22 and 23 only.  Set bit 29 (TB PAR
;12962  ;           DIAG) and bit 2 (MME) in CSR 1 to force an error sum on the WRITE.V
;12963  ;           .NOCHK operation.
;12964  ;       3)  Clear WR 1 (expected data) and clear the CURRENT MODE bits in the
;12965  ;           PSL (Current mode=Kernel).
;12966  ;       4)  Write the TB with 0's in the PROT and MODIFY bits.
;12967  ;       5)  Issue MEM.REQ with MF=WRITE.V.NOCHK and DT=longword.  Issue a
;12968  ;           WRITE.MEM LS to complete the WRITE.V.NOCHK function.
;12969  ;       6)  Read CSR 1 and check bits 22 and 23 (ACCESS REFUSED and MODIFY
;12970  ;           REFUSED for 0.
;12971  ;       7)  Increment the data in the TB PROT and MODIFY bits and repeat steps
;12972  ;           5 and 6.  Repeat until PROT and MODIFY bits are all 1.
;12973  ;       8)  Increment CURRENT MODE bits of PSL and repeat steps 3 through 7.
;12974  ;           Repeat until remaining modes have all been tested (Executive, Sup-
;12975  ;           ervisor, and user).
;12976  ;       9)  Clear the CURRENT MODE bits in the PSL (Current mode=Kernel).
;12977  ;       10) Write the TB with 0's in the PROT and MODIFY bits.
;12978  ;       11) Calculate the expected data by reading LS[ACCESS.RCHK.KERNEL] and
;12979  ;           LS[MODIFY.RCHK.KERNAL].  Save these in WR[2] and WR[3].  The LSB
;12980  ;           of WR[2] and WR[3] is the expected data for ACCESS REFUSED and
;12981  ;           MODIFY REFUSED respectively.  Load the expected data into WR[1].
;12982  ;       12) Issue MEM.REQ with MF=TEST.V.RCHK and DT=longword.  Issue a MOV
;12983  ;           MEM.DATA to WR[0] to complete the TEST.V.RCHK function, but there is
;12984  ;           no need to check this information.
;12985  ;       13) Read CSR 1 and check bits 22 and 23 (ACCESS REFUSED and MODIFY
;12986  ;           REFUSED for the correct data.
;12987  ;       14) Increment the data in the TB PROT and MODIFY bits.  Shift WR[2] and
;12988  ;           WR[3] right one bit to get the next expected data into the LSB.
;12989  ;           Repeat steps 11 through 13.  Repeat until PROT and MODIFY bits are
;12990  ;           all 1.
;12991  ;       15) Increment CURRENT MODE bits of PSL and repeat steps 10 through 14.
;12992  ;           Use LS[ACCESS.RCHK.EXEC] and LS[MODIFY.RCHK.EXEC] in step 11 for
;12993  ;           executive mode.  Use LS[ACCESS.RCHK.SUP] and LS[MODIFY.RCHK.SUP]
;12994  ;           for supervisor mode.  Use LS[ACCESS.RCHK.USER] and LS[MODIFY.RCK
;12995  ;           .USER] for user mode.  Repeat until all modes have all been tested.
;12996  ;       16) Repeat steps 9 through 15 with WCHK substituted for RCHK.
;12997  ;
;12998  ; Errors:
;12999  ;
;13000  ;       Note 1:  The address of the PROM being accessed will be printed under
;13001  ;       OTHER DATA in the error report, so that address lines can be checked
;13002  ;       in debug.  The address printed uses PROT A as the LSB (address bit 0)
;13003  ;       and L CM1 as the MSB (address bit 8).  This is the reverse of the
;13004  ;       address lines on the board but corresponds to the PROM listing.
;13005  ;
;13006  ;       NOTE 2:  Expected and received data is CSR 1 bits 22 and 23 (ACC or MOD
;13007  ;                refused).
;13008  ;
;13009  ;       Error 1 - ACCESS REFUSED and/or MODIFY REFUSED set on NOCHK operation.
;13010  ;       Error 2 - ACCESS REFUSED and/or MODIFY REFUSED incorrect on RCHK oper-
;13011  ;                 ation.  Kernal mode.
;13012  ;       Error 3 - ACCESS REFUSED and/or MODIFY REFUSED incorrect on RCHK oper-
;13013  ;                 ation.  Executive mode.
;13014  ;       Error 4 - ACCESS REFUSED and/or MODIFY REFUSED incorrect on RCHK oper-
;13015  ;                 ation.  Supervisor mode.
```

```
;13016  ;       Error 5 - ACCESS REFUSED and/or MODIFY REFUSED incorrect on RCHK oper-
;13017  ;                 ation.  User mode.
;13018  ;       Error 6 - ACCESS REFUSED and/or MODIFY REFUSED incorrect on WCHK oper-
;13019  ;                 ation.  Kernal mode.
;13020  ;       Error 7 - ACCESS REFUSED and/or MODIFY REFUSED incorrect on WCHK oper-
;13021  ;                 ation.  Executive mode.
;13022  ;       Error 8 - ACCESS REFUSED and/or MODIFY REFUSED incorrect on WCHK oper-
;13023  ;                 ation.  Supervisor mode.
;13024  ;       Error 9 - ACCESS REFUSED and/or MODIFY REFUSED incorrect on WCHK oper-
;13025  ;                 ation.  User mode.
;13026  ;
;13027  ; Debug:
;13028  ;
;13029  ;       Two methods of debug will be explained.  If the MCT module is in a
;13030  ;       test station then the MCT can be single stepped making debug easier
;13031  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;13032  ;       method will be explained also.  When single stepping the memory con-
;13033  ;       troller will help in debug, this section will explicitly suggest it.
;13034  ;
;13035  ;       Error 1 - This error iNdicates a problem with the protection PROM, its
;13036  ;       MF inputs, or the CSR logic.  Single step the CPU to the MEM.REQ that
;13037  ;       issues WRITE.V.NOCHK.  The memory u-code will be looping waiting for
;13038  ;       CPU DATA REQ to be asserted.  Check the outputs ACCESSED REFUSED L and
;13039  ;       MODIFY REFUSED L of the protection PROM for highs.  If these are in-
;13040  ;       correct, check the inputs for a low on both L MF0 H and L MF1 H.  Also
;13041  ;       check the enable on pin 13 for a ground connection.  If these inputs
;13042  ;       are correct, the protection PROM is probably bad.  If one of the MF
;13043  ;       inputs is high, check them at the output of the latch with CSR 07 H
;13044  ;       and CSR 08 H as the inputs.  These inputs should both be low.
;13045  ;         If ACCESSED REFUSED L and MODIFY REFUSED L are high at the protection
;13046  ;       PROM, then check them at the inputs of the CSR 1A PAL.  If they are
;13047  ;       high at the CSR 1A PAL, then the PAL is probably bad.
;13048  ;
;13049  ;       Error 2 - This error indicates a problem with the Protection PROM, its
;13050  ;       inputs, or the CSR 1A PAL.  Single step the CPU to the MEM.REQ that
;13051  ;       issues TEST.V.RCHK.  The memory u-code will loop waiting for CPU GRANT
;13052  ;       to go away.  The Protection PROM can be scoped at this time.  Check
;13053  ;       the ouputs for the expected result first (under EXPECTED DATA bits 22
;13054  ;       and 23).  If these are incorrect, check the inputs for the correct ad-
;13055  ;       dress on the PROT and MODIFY lines.  The other lines are as follows:
;13056  ;       L CM0 H, L CM1 H, and L MF0 H should all be low.  L MF1 H should be
;13057  ;       high.  The enable on pin 13 should be low.  If all these are correct,
;13058  ;       the PROM itself is probably bad.  The MF and CM inputs can be checked
;13059  ;       at the latch which supplies them.  The inputs to the latch should also
;13060  ;       be in the correct state (same as the outputs).
;13061  ;         If the outputs are correct, check them at the input to the CSR 1A
;13062  ;       PAL.  If correct there, the PAL is probably bad.
;13063  ;
;13064  ;       Error 3 - This error indicates a problem with the Protection PROM or
;13065  ;       its inputs. Single step the CPU to the MEM.REQ that issues TEST.V.RCHK.
;13066  ;       The memory u-code will loop and the protection PROM can be checked.
;13067  ;       Check the inputs for the correct address on the PROT and MODIFY lines.
;13068  ;       The other inputs are as follows:  L CM1 H and L MF0 H should be low.
;13069  ;       L CM0 H and L MF1 H should be high. The MF and CM inputs can be checked
;13070  ;       at the latch which supplies them.  The inputs to the latch should also
```

```
                ;13071  ;       be in the correct state (same as the outputs).  If the inputs are cor-
                ;13072  ;       rect, suspect the PROM itself.
                ;13073  ;
                ;13074  ;       Error 4 - Same as error 3 except L CM1 H should be high and L CM0 H
                ;13075  ;       should be low.
                ;13076  ;
                ;13077  ;       Error 5 - Same as error 3 except L CM1 H should be high.
                ;13078  ;
                ;13079  ;       Error 6 - This error indicates a problem with the Protection PROM, its
                ;13080  ;       inputs, or the CSR 1A PAL.  Single step the CPU to the MEM.REQ that
                ;13081  ;       issues TEST.V.WCHK.  The memory u-code will loop waiting for CPU GRANT
                ;13082  ;       to go away.  The Protection PROM can be scoped at this time.  Check
                ;13083  ;       the ouputs for the expected result first (under EXPECTED DATA bits 22
                ;13084  ;       and 23).  If these are incorrect, check the inputs for the correct ad-
                ;13085  ;       dress on the PROT and MODIFY lines.  The other lines are as follows:
                ;13086  ;       L CM0 H, L CM1 H, and L MF1 H should all be low.  L MF0 H should be
                ;13087  ;       high.  The enable on pin 13 should be low.  If all these are correct,
                ;13088  ;       the PROM itself is probably bad.  The MF and CM inputs can be checked
                ;13089  ;       at the latch which supplies them.  The inputs to the latch should also
                ;13090  ;       be in the correct state (same as the outputs).
                ;13091  ;         If the outputs are correct, check them at the input to the CSR 1A
                ;13092  ;       PAL.  If correct there, the PAL is probably bad.
                ;13093  ;
                ;13094  ;       Errors 7 through 9 - The protection PROM itself is the most likely sus-
                ;13095  ;       pect.
                ;13096  ;
                ;13097  ;--
                ;13098
                ;13099  T.19:
                ;13100
U 0EF8, B65E,15 ;13101          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;13102                                          ; STATUS WORD
U 0EF9, 3E80,15 ;13103          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;13104                                          ; BIT 15 INDICATES START OF TEST TO
                ;13105                                          ; CONSOLE
U 0EFA, 10E0,15 ;13106          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;13107  WAIT.T19.0:
U 0EFB, 88EF,B4 ;13108          JMP [WAIT.T19.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;13109                                          ; RESPOND
                ;13110
U 0EFC, 0A1A,AC ;13111          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                ;13112
U 0EFD, 5F6C,15 ;13113          MCOM LS[ACCESS.REFUSED] TO WR[0] ;CLEAR BIT 22 IN WR 0
U 0EFE, 456E,15 ;13114          BIC LS[MODIFY.REFUSED] TO WR[0] ;ALSO CLEAR BIT 23
U 0EFF, 3E8A,15 ;13115          MOV WR[0] TO LS[ERROR.MASK]     ;SET UP ERROR MASK TO CHECK BITS 22
                ;13116                                          ; AND 23 ONLY (ACCESS AND MODIFY REFUSED)
U 0F00, B670,15 ;13117          MOV LS[OTHER.DATA] TO WR[0]     ;SET BIT 24 IN WR
U 0F01, 3E80,15 ;13118          MOV WR[0] TO LS[CONTROL.STATUS] ;ENABLE PRINTOUT UNDER 'OTHER' IN ERROR
                ;13119                                          ; REPORT (PROM ADDRESS ACCESSED)
U 0F02, B676,15 ;13120          MOV LS[MME] TO WR[0]            ;SET BIT 27 IN WR 0
U 0F03, C77A,15 ;13121          BIS LS[TB.PAR.DIAG] TO WR[0]    ;ALSO SET BIT 29
U 0F04, 8A17,FC ;13122          JSR [WRITE.CSR1]                ;ENABLE MEMORY MANAGEMENT AND TB PARITY
                ;13123                                          ; ERROR
U 0F05, 6514,15 ;13124          CLR LS[T10]                     ;DATA FOR WRITE TO TB
U 0F06, 2F80,15 ;13125          CLR WR[0]                       ;CLEAR WR 0 FOR MODE (CURRENT MODE=KERNAL)
```

```
U 0F07, E516,15 ;13126          CLR LS[T11]                     ;SAVE CURRENT MODE SETTING
                ;13127  REPEAT.T19.1.NEXTMODE:
U 0F08, 88F5,4C ;13128          JSR [T19.PROM.ADDRESS]          ;GO TO CALCULATE PROM ADDRESS TO PRINT
                ;13129  REPEAT.T19.1:
U 0F09, 989C,F5 ;13130          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;13131                                          ;SET UP TO WRITE TB WITH CURRENT PROT
                ;13132                                          ; AND MODIFY VALUE
U 0F0A, B214,15 ;13133          WRITE.MEM LS[T10]               ;WRITE DATA IN TB
                ;13134  LOOP.T19.1:
U 0F0B, 2F82,95 ;13135          CLR WR[1]                       ;EXPECTED DATA
U 0F0C, 1B9C,75 ;13136          MEM.REQ[WRITE.V.NOCHK] ADRS[#0] DT[LONG]
                ;13137                                          ;SET UP TO CLOCK CSR 1 WITH ACCESS INFO
U 0F0D, B29C,15 ;13138          WRITE.MEM LS[ZERO]              ;ISSUE WRITE TO MEMORY
U 0F0E, 9D45,75 ;13139          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;13140                                          ;SET UP TO READ CSR 1
U 0F0F, 3022,15 ;13141          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 0F10, 0869,3C ;13142          JSR [CHECK.RESULT]              ;CHECK FOR ACCESS REFUSED AND MODIFY
                ;13143                                          ; REFUSED BOTH CLEAR
U 0F11, 08F0,B4 ;13144          JMP [LOOP.T19.1]                ;ERROR LOOP IF ENABLED
                ;13145
U 0F12, 88F6,8C ;13146          JSR [T19.INC.PROT]              ;CALCULATE NEXT PROT AND MODIFY SETTING
U 0F13, 88F0,94 ;13147          JMP [REPEAT.T19.1]              ;SUBROUTINE WILL RETURN HERE IF NOT DONE
                ;13148                                          ; ALL PROT AND MODIFY YET
                ;13149
U 0F14, B616,15 ;13150          MOV LS[T11] TO WR[0]            ;ELSE GET CURRENT MODE SETTING
U 0F15, 4070,15 ;13151          ADD LS[BIT24] TO WR[0]          ;INCREMENT MODE VALUE
U 0F16, 3E16,15 ;13152          MOV WR[0] TO LS[T11]            ;STORE MODE IN LS
                ;13153          BIT LS[BIT26] WITH WR[0],       ;SEE IF BIT 26 SET (MEANS DONE ALL MODES)
U 0F17, D974,35 ;13154           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0F18, 88F0,89 ;13155          JMP.IF[BITS.CLR] TO [REPEAT.T19.1.NEXTMODE] ;REPEAT IF NOT DONE ALL
                ;13156                                          ; MODES YET
                ;13157
U 0F19, E5F8,15 ;13158          CLR LS[OS]                      ;CLEAR INDEX
U 0F1A, FDF8,15 ;13159          COM LS[OS]                      ;COMPLEMENT INDEX.  NOW WHEN INDEX INC-
                ;13160                                          ; REMENTED FIRST TIME, WILL GO TO 0
U 0F1B, 2F80,15 ;13161          CLR WR[0]                       ;CLEAR WR 0 FOR MODE (CURRENT MODE=KERNAL)
U 0F1C, E516,15 ;13162          CLR LS[T11]                     ;SAVE CURRENT MODE SETTING
                ;13163  REPEAT.T19.2.NEXTMODE:
U 0F1D, FF82,15 ;13164          INC LS[ERROR.NUMBER]            ;ERRORS 2 THROUGH 5
U 0F1E, 6514,15 ;13165          CLR LS[T10]                     ;DATA FOR WRITE TO TB
U 0F1F, 88F5,4C ;13166          JSR [T19.PROM.ADDRESS]          ;GO TO CALCULATE PROM ADDRESS TO PRINT
U 0F20, B64C,15 ;13167          MOV LS[BIT6] TO WR[0]           ;SET BIT 6 IN WR
U 0F21, 6F88,15 ;13168          XOR WR[0] TO LS[ADDRESS.DATA]   ;SET BIT 6 IN ADDRESS TO PRINT
U 0F22, 7FF8,15 ;13169          INC LS[OS]                      ;POINT TO NEXT ACCESS DATA PATTERN
U 0F23, 37F7,15 ;13170          MOV LS[USER.INDEX(4-0)] TO WR[2] ;LOAD WR 2 WITH EXPECTED ACCESS REFUSED
                ;13171                                          ; DATA STORED IN 2ND HALF OF INDEXED
                ;13172                                          ; PORTION OF LS
U 0F24, 7FF8,15 ;13173          INC LS[OS]                      ;INCREMENT INDEX
U 0F25, B7F7,95 ;13174          MOV LS[USER.INDEX(4-0)] TO WR[3] ;LOAD WR 3 WITH EXPECTED MODIFY REFUSED
                ;13175                                          ; DATA STORED IN NEXT LS LOCATION
                ;13176  REPEAT.T19.2:
U 0F26, 989C,F5 ;13177          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;13178                                          ;SET UP TO WRITE TB WITH CURRENT PROT
                ;13179                                          ; AND MODIFY VALUE
U 0F27, B214,15 ;13180          WRITE.MEM LS[T10]               ;WRITE DATA IN TB
```

```
U 0F28, 88F5,EC ;13181          JSR [T19.COMPUTE.RESULT]        ;COMPUTE EXPECTED DATA
                ;13182  LOOP.T19.2:
U 0F29, B618,95 ;13183          MOV LS[T12] TO WR[1]            ;EXPECTED DATA
                ;13184
U 0F2A, 989D,75 ;13185          MEM.REQ[TEST.V.RCHK] ADRS[#0] DT[LONG]
                ;13186                                          ;SET UP TO CLOCK CSR 1 WITH ACCESS INFO
U 0F2B, 3022,15 ;13187          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE MEMORY CYCLE (NO
                ;13188                                          ; NEED TO CHECK THIS DATA)
U 0F2C, 9D45,75 ;13189          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;13190                                          ;SET UP TO READ CSR 1
U 0F2D, 3022,15 ;13191          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 0F2E, 0869,3C ;13192          JSR [CHECK.RESULT]              ;CHECK ACCESS REFUSED AND MODIFY
                ;13193                                          ; REFUSED FOR CORRECT DATA
U 0F2F, 08F2,94 ;13194          JMP [LOOP.T19.2]                ;ERROR LOOP IF ENABLED
                ;13195
U 0F30, 88F6,8C ;13196          JSR [T19.INC.PROT]              ;CALCULATE NEXT PROT AND MODIFY SETTING
U 0F31, 08F2,64 ;13197          JMP [REPEAT.T19.2]              ;SUBROUTINE WILL RETURN HERE IF NOT DONE
                ;13198                                          ; ALL PROT AND MODIFY YET
                ;13199
U 0F32, B616,15 ;13200          MOV LS[T11] TO WR[0]            ;ELSE GET CURRENT MODE SETTING
U 0F33, 4070,15 ;13201          ADD LS[BIT24] TO WR[0]          ;INCREMENT MODE VALUE
U 0F34, 3E16,15 ;13202          MOV WR[0] TO LS[T11]            ;STORE MODE IN LS
                ;13203          BIT LS[BIT26] WITH WR[0],       ;SEE IF BIT 26 SET (MEANS DONE ALL MODES)
U 0F35, D974,35 ;13204           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0F36, 08F1,D9 ;13205          JMP.IF[BITS.CLR] TO [REPEAT.T19.2.NEXTMODE] ;REPEAT IF NOT DONE ALL
                ;13206                                          ; MODES YET
                ;13207
U 0F37, 2F80,15 ;13208          CLR WR[0]                       ;CLEAR WR 0 FOR MODE (CURRENT MODE=KERNAL)
U 0F38, E516,15 ;13209          CLR LS[T11]                     ;SAVE CURRENT MODE SETTING
                ;13210  REPEAT.T19.6.NEXTMODE:
U 0F39, FF82,15 ;13211          INC LS[ERROR.NUMBER]            ;ERRORS 6 THROUGH 9
U 0F3A, 6514,15 ;13212          CLR LS[T10]                     ;DATA FOR WRITE TO TB
U 0F3B, 88F5,4C ;13213          JSR [T19.PROM.ADDRESS]          ;GO TO CALCULATE PROM ADDRESS TO PRINT
U 0F3C, B64A,15 ;13214          MOV LS[BIT5] TO WR[0]           ;SET BIT 5 IN WR
U 0F3D, 6F88,15 ;13215          XOR WR[0] TO LS[ADDRESS.DATA]   ;SET BIT 5 IN ADDRESS TO PRINT
U 0F3E, 7FF8,15 ;13216          INC LS[OS]                      ;POINT TO NEXT ACCESS DATA PATTERN
U 0F3F, 37F7,15 ;13217          MOV LS[USER.INDEX(4-0)] TO WR[2] ;LOAD WR 2 WITH EXPECTED ACCESS REFUSED
                ;13218                                          ; DATA STORED IN 2ND HALF OF INDEXED
                ;13219                                          ; PORTION OF LS
U 0F40, 7FF8,15 ;13220          INC LS[OS]                      ;INCREMENT INDEX
U 0F41, B7F7,95 ;13221          MOV LS[USER.INDEX(4-0)] TO WR[3] ;LOAD WR 3 WITH EXPECTED MODIFY REFUSED
                ;13222                                          ; DATA STORED IN NEXT LS LOCATION
                ;13223  REPEAT.T19.6:
U 0F42, 989C,F5 ;13224          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;13225                                          ;SET UP TO WRITE TB WITH CURRENT PROT
                ;13226                                          ; AND MODIFY VALUE
U 0F43, B214,15 ;13227          WRITE.MEM LS[T10]               ;WRITE DATA IN TB
U 0F44, 88F5,EC ;13228          JSR [T19.COMPUTE.RESULT]        ;COMPUTE EXPECTED DATA
                ;13229  LOOP.T19.6:
U 0F45, B618,95 ;13230          MOV LS[T12] TO WR[1]            ;EXPECTED DATA
                ;13231
U 0F46, 199C,F5 ;13232          MEM.REQ[TEST.V.WCHK] ADRS[#0] DT[LONG]
                ;13233                                          ;SET UP TO CLOCK CSR 1 WITH ACCESS INFO
U 0F47, 3022,15 ;13234          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE MEMORY CYCLE (NO
                ;13235                                          ; NEED TO CHECK THIS DATA)
```

```
U 0F48, 9D45,75 ;13236          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;13237                                  ;SET UP TO READ CSR 1
U 0F49, 3022,15 ;13238          MOV MEM.DATA TO WR[0]    ;GET CONTENTS OF CSR 1
U 0F4A, 0869,3C ;13239          JSR [CHECK.RESULT]       ;CHECK ACCESS REFUSED AND MODIFY
                ;13240                                  ; REFUSED FOR CORRECT DATA
U 0F4B, 08F4,54 ;13241          JMP [LOOP.T19.6]         ;ERROR LOOP IF ENABLED
                ;13242
U 0F4C, 88F6,8C ;13243          JSR [T19.INC.PROT]       ;CALCULATE NEXT PROT AND MODIFY SETTING
U 0F4D, 88F4,24 ;13244          JMP [REPEAT.T19.6]       ;SUBROUTINE WILL RETURN HERE IF NOT DONE
                ;13245                                  ; ALL PROT AND MODIFY YET
                ;13246
U 0F4E, B616,15 ;13247          MOV LS[T11] TO WR[0]     ;ELSE GET CURRENT MODE SETTING
U 0F4F, 4070,15 ;13248          ADD LS[BIT24] TO WR[0]   ;INCREMENT MODE VALUE
U 0F50, 3E16,15 ;13249          MOV WR[0] TO LS[T11]     ;STORE MODE IN LS
                ;13250          BIT LS[BIT26] WITH WR[0], ;SEE IF BIT 26 SET (MEANS DONE ALL MODES)
U 0F51, D974,35 ;13251           DT(LONG)&SET.ALU.CC     ; AND SET CONDITION CODES
U 0F52, 08F3,99 ;13252          JMP.IF[BITS.CLR] TO [REPEAT.T19.6.NEXTMODE] ;REPEAT IF NOT DONE ALL
                ;13253                                  ; MODES YET
                ;13254
U 0F53, 88F7,24 ;13255          JMP [END.T19]            ;DONE
                ;13256
                ;13257
                ;13258  T19.PROM.ADDRESS:
U 0F54, BFFE,15 ;13259          MOV WR[0] TO LS[PSL.HW]  ;SET UP PSL BITS IN CPU (CURRENT MODE)
U 0F55, 2F80,15 ;13260          CLR WR[0]                ;START WITH ADDRESS AT 0
U 0F56, 3616,95 ;13261          MOV LS[T11] TO WR[1]     ;GET CURRENT MODE
                ;13262          BIT LS[BIT24] WITH WR[1], ;SEE IF CM0 IS SET
U 0F57, D970,B5 ;13263           DT(LONG)&SET.ALU.CC     ; AND SET CONDITION CODES
U 0F58, 5B00,09 ;13264          SKIP.IF[BITS.CLR]        ;SKIP NEXT INSTRUCTION IF CM0 CLEAR
U 0F59, 474E,15 ;13265          BIS LS[BIT7] TO WR[0]    ;ELSE SET ADDRESS BIT 7
                ;13266          BIT LS[BIT25] WITH WR[1], ;SEE IF CM1 IS SET
U 0F5A, 5972,B5 ;13267           DT(LONG)&SET.ALU.CC     ; AND SET CONDITION CODES
U 0F5B, 5B00,09 ;13268          SKIP.IF[BITS.CLR]        ;SKIP NEXT INSTRUCTION IF CM1 CLEAR
U 0F5C, 4750,15 ;13269          BIS LS[BIT8] TO WR[0]    ;ELSE SET ADDRESS BIT 8
                ;13270          MOV WR[0] TO LS[ADDRESS.DATA], ;STORE RESULT IN LS FOR PRINTOUT IF ERROR
U 0F5D, 3E88,14 ;13271           RETURN                  ; RETURN TO MAIN CODE (MF0 AND MF1, AD-
                ;13272                                  ; DRESS BITS 5 AND 6, ARE SET UP IN
                ;13273                                  ; MAIN CODE)
                ;13274
                ;13275  T19.COMPUTE.RESULT:
U 0F5E, 2F82,95 ;13276          CLR WR[1]                ;START WITH CLEAR EXPECTED DATA
                ;13277          BIT LS[BIT0] WITH WR[2], ;SEE IF ACCESS SHOULD BE 1 OR 0
U 0F5F, D941,35 ;13278           DT(LONG)&SET.ALU.CC     ; AND SET CONDITION CODES
U 0F60, 5B00,09 ;13279          SKIP.IF[BITS.CLR]        ;SKIP NEXT INSTRUCTION IF ACCESS BIT
                ;13280                                  ; IS CLEAR
U 0F61, C76C,95 ;13281          BIS LS[ACCESS.REFUSED] TO WR[1] ;ELSE SET BIT 22 IN EXPECTED DATA
                ;13282          BIT LS[BIT0] WITH WR[3], ;SEE IF MODIFY SHOULD BE 1 OR 0
U 0F62, 5941,B5 ;13283           DT(LONG)&SET.ALU.CC     ; AND SET CONDITION CODES
U 0F63, 5B00,09 ;13284          SKIP.IF[BITS.CLR]        ;SKIP NEXT INSTRUCTION IF MODIFY BIT
                ;13285                                  ; IS CLEAR
U 0F64, 476E,95 ;13286          BIS LS[MODIFY.REFUSED] TO WR[1] ;ELSE SET BIT 23 IN EXPECTED DATA
U 0F65, 3E18,95 ;13287          MOV WR[1] TO LS[T12]     ;SAVE IN LS
U 0F66, A341,15 ;13288          ROR WR[2]                ;PUT NEXT ACCESS DATA BIT IN POSITION 0
                ;13289          ROR WR[3],               ;PUT NEXT MODIFY DATA BIT IN POSITION 0
U 0F67, A341,94 ;13290           RETURN                  ; GO BACK TO MAIN PROGRAM
```

```
                ;13291
                ;13292
                ;13293  T19.INC.PROT:
U 0F68, FF88,15 ;13294          INC LS[ADDRESS.DATA]            ;INCREMENT ADDRESS TO PRINT IF ERROR
U 0F69, B676,15 ;13295          MOV LS[BIT27] TO WR[0]          ;SET UP TO INCREMENT DATA IN TB (PROT
                ;13296                                          ; BITS)
                ;13297          ADD WR[0] TO LS[T10],           ;INCREMENT TB DATA PROT BITS
U 0F6A, EC14,35 ;13298           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 0F6B, 88F7,00 ;13299          JMP.IF[N.CLR] TO [T19.INC.RETURN] ;REPEAT IF BIT 31 (VALID) HAS NOT YET SET
                ;13300                                          ; ELSE SEE IF MODIFY WAS SET
U 0F6C, 3674,15 ;13301          MOV LS[BIT26] TO WR[0]          ;SET BIT 26 IN WR
                ;13302          BIT WR[0] WITH LS[T10],         ;SEE IF MODIFY BIT WAS SET
U 0F6D, D914,35 ;13303           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 0F6E, 88F7,11 ;13304          JMP.IF[BIT.SET] TO [T19.INC.RETURN+1] ;GO DO NEXT MODE IF MODIFY WAS SET
U 0F6F, BE14,15 ;13305          MOV WR[0] TO LS[T10]            ;ELSE SET MODIFY BIT IN TB DATA
                ;13306  T19.INC.RETURN:
U 0F70, 5B00,14 ;13307          RETURN                          ;AND CONTINUE
                ;13308  T19.INC.RETURN+1:
U 0F71, DB00,16 ;13309          RETURN+1                        ;DO NEXT MODE
                ;13310
                ;13311  END.T19:
```

```
;13312  .PAGE   " Other CSR 1 Bits Test"
;13313
;13314  .toc    "  TEST 1A NXM And Adapter Reg Select Test (M8391  MCT Module)"
;13315  ;++
;13316  ;
;13317  ; Test Description:
;13318  ;
;13319  ;       This test checks the NXM bit by attempting a write at address 0, and
;13320  ;       writes at non-existant addresses.  The first write should complete
;13321  ;       without an NXM while the other writes should receive an NXM.  The ADAP
;13322  ;       REG SEL bit (bit 18) is also checked to assure that it comes up on a
;13323  ;       write to the UNIBUS Adapter Register address space.
;13324  ;         The data path is as follows:  An LS location is set up with an ad-
;13325  ;       dress and a MEM.REQ is issued with WRITE.P (write physical) as the
;13326  ;       memory function and DT=longword.  A WRITE.MEM LS[ZEROS] is executed to
;13327  ;       complete the memory cycle and CSR 1 is read to check the NXM bit (bit
;13328  ;       16) and the ADAP REG SEL bit (bit 18).  It is not necessary to describe
;13329  ;       the micro-flow of the WRITE.P function as all that is needed is a CSR
;13330  ;       ERR SUM CLOCK.  This much was tested in the protection PROM test.  The
;13331  ;       PA bits 18-23 sent over as the address on the MEM.REQ goes to two PALs
;13332  ;       along with the finger print signals from the array board(s) to produce
;13333  ;       the correct state of the NXM bit (NXM L).  This goes to the CSR 1A PAL
;13334  ;       and is clocked in by the micro code.  The signal UB ADAPTER REG SEL L
;13335  ;       is generated by the DECODER B PAL according to the value of PA 23-18
;13336  ;       and sent to the CSR 1B PAL.  The micro code will also clock the value
;13337  ;       of this bit into the CSR.
;13338  ;
;13339  ; Assumptions:
;13340  ;
;13341  ;       It is assumed that all previous tests have run successfully.
;13342  ;
;13343  ; Test Steps:
;13344  ;
;13345  ;       1)  Set up error number and module number in LS (for error printouts)
;13346  ;           and clear previous error number in LS.
;13347  ;       2)  Set up error mask to check bits 16 (NXM) and 18 (UB ADAP REG) only.
;13348  ;       3)  Execute MEM.REQ with MF=WRITE.P and DT=longword.  Use an LS loc-
;13349  ;           ation containing all 0's as the address.
;13350  ;       4)  Execute WRITE.MEM LS with data of zeros.  Then read the CSR and
;13351  ;           check for NXM clear and UB ADAP REG SEL clear.
;13352  ;       5)  Load an LS location with the value 540000(H).  This is just above
;13353  ;           the highest memory address possible in the NEBULA system.
;13354  ;       6)  Execute MEM.REQ with MF=WRITE.P and DT=longword.  Use the LS loc-
;13355  ;           ation just loaded as the physical address.
;13356  ;       7)  Execute WRITE.MEM LS with data of 0's.  Then read CSR 1 and check
;13357  ;           for NXM set and UB ADAP REG SEL clear.
;13358  ;       8)  Increment the LS location containing the physical address by 40000
;13359  ;           (H).  Repeat steps 6 through 8 until address = F00000(H).
;13360  ;       9)  Repeat steps 6 through 8 checking for NXM clear and UB ADAP REG SEL
;13361  ;           set until address = FC0000(H).
;13362  ;       10) Repeat steps 6 and 7 checking for NXM clear and UB ADAP REG SEL
;13363  ;           clear at address FFFFFF(H).
;13364  ;
;13365  ; Errors:
;13366  ;
```

```
;13367  ;       NOTE:  Expected and received data is CSR 1 bits 18 and 16 (UB ADP REG
;13368  ;              and NXM).
;13369  ;
;13370  ;       Error 1 - NXM failed to clear or UB ADAP REG SEL failed to clear on
;13371  ;                 write physical to existent memory (address 0).
;13372  ;       Error 2 - NXM failed to set or UB ADAP REG SEL failed to clear on
;13373  ;                 write physical to non-existent memory (address =540000(H)
;13374  ;                 through EC0000(H))
;13375  ;       Error 3 - NXM failed to clear or UB ADAP REG SEL failed to set on write
;13376  ;                 physical to UNIBUS ADAPTER REG (address =F00000(H) through
;13377  ;                 F80000(H)).  Address under OTHER.
;13378  ;       Error 4 - NXM failed to clear or UB ADAP REG SEL failed to clear on
;13379  ;                 write physical to UNIBUS address space (address FFFFFF(H)).
;13380  ;                 Address under OTHER.
;13381  ;
;13382  ; Debug:
;13383  ;
;13384  ;       Note:  The address accessed when an error occurs will be printed under
;13385  ;       OTHER in the error report.
;13386  ;
;13387  ;       Two methods of debug will be explained.  If the MCT module is in a
;13388  ;       test station then the MCT can be single stepped making debug easier
;13389  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;13390  ;       method will be explained also.  When single stepping the memory con-
;13391  ;       troller will help in debug, this section will explicitly suggest it.
;13392  ;
;13393  ;       Error 1 - This error indicates a problem with the DECODER A PAL, the
;13394  ;       DECODER B PAL, the CSR 1A PAL, the CSR 1B PAL or their inputs.  The
;13395  ;       inputs to the decoder PALs can be checked by single stepping the CPU
;13396  ;       to the MEM.REQ instruction with MF=WRITE.P.  The memory micro-code will
;13397  ;       loop waiting for CPU DATA REQ.  The output P NXM L from DECODER A
;13398  ;       should be high.  If not, check the inputs to DECODER A for lows on PA
;13399  ;       23 through PA 18 and a low on FP3A L.  If these inputs are OK, DECODER
;13400  ;       A is bad.
;13401  ;         If P NXM L from the DECODER A PAL is high, then DECODER A is OK.
;13402  ;       Check the outputs from DECODER B for a high on NXM L and UB ADAPTER REG
;13403  ;       SEL L.  If either signal is low, check the inputs for a high on P NXM L
;13404  ;       and lows on PA 23 H through PA 18 H.  If these inputs are OK, the
;13405  ;       DECODER B PAL is bad.
;13406  ;         If the outputs NXM L and UB ADAPTER REG SEL L are both high, check
;13407  ;       them at the inputs to the CSR 1A and CSR 1B PALs.  If they are OK there
;13408  ;       the CSR 1 PAL is bad.
;13409  ;
;13410  ;       Error 2 - This error indicates a problem with the decoder PALs or the
;13411  ;       CSR logic.  The inputs to the decoder PALs can be checked by single
;13412  ;       stepping the CPU to the MEM.REQ instruction with MF=WRITE.P.  The
;13413  ;       memory micro-code will loop waiting for CPU DATA REQ.  The output P NXM
;13414  ;       L from the DECODER A PAL should be low.  If not, check the inputs PA 23
;13415  ;       H through PA 18 H for a 010101(B) respectively or higher binary value
;13416  ;       (the actual address is shown under OTHER DATA in the printout, but any
;13417  ;       value greater than 010101 in PA 23 - PA 18 should produce an NXM).  If
;13418  ;       these inputs are OK, suspect the DECODER A PAL.
;13419  ;         If P NXM L is low, check the output NXM L and UB ADAPTER REG SEL L
;13420  ;       from the DECODER B PAL.  NXM L should be low and UB ADAPTER REG SEL L
;13421  ;       should be high.  If either of these outputs is bad, check the inputs to
```

```
                    ;13422  ;       the DECODER B PAL for a value of between 010101(B) and 111011(B) on PA
                    ;13423  ;       23 H through PA 18 H respectively.  The actual address is shown under
                    ;13424  ;       OTHER DATA in the printout.  Also check that the P NXM L input is low.
                    ;13425  ;       If these are OK, the PAL is bad.
                    ;13426  ;         If NXM L and UB ADAPTER REG SEL L are OK, then check them at the CSR
                    ;13427  ;       PAL inputs.  The CSR PAL is bad if these inputs are OK.
                    ;13428  ;
                    ;13429  ;       Error 3 - Follow same procedure as error 2 except P NXM and NXM L
                    ;13430  ;       should be high and UB ADAPTER REG SEL L should be low.  The inputs PA
                    ;13431  ;       23 - PA 18 should be between 111100(B) and 111110(B).
                    ;13432  ;
                    ;13433  ;       Error 4 - Follow same procedure as error 2 except P NXM L and NXM L
                    ;13434  ;       should be high and the address on PA 23 - PA 18 should be 111111(B).
                    ;13435  ;       One of the decoder PALs is the most likely suspect.
                    ;13436  ;
                    ;13437  ;--
                    ;13438
                    ;13439  T.1A:
                    ;13440
U 0F72, B65E,15     ;13441          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;13442                                          ; STATUS WORD
U 0F73, 3E80,15     ;13443          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;13444                                          ; BIT 15 INDICATES START OF TEST TO
                    ;13445                                          ; CONSOLE
U 0F74, 10E0,15     ;13446          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;13447  WAIT.T1A.0:
U 0F75, 08F7,54     ;13448          JMP [WAIT.T1A.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;13449                                          ; RESPOND
                    ;13450
U 0F76, 0A1A,9C     ;13451          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                    ;13452                                          ; CLEAR MCT CSR 1
U 0F77, 5F60,15     ;13453          MCOM LS[NXM] TO WR[0]           ;CLEAR BIT 16 IN WR
U 0F78, 4564,15     ;13454          BIC LS[ADP.REG] TO WR[0]        ;CLEAR BIT 18 IN WR
U 0F79, 3E8A,15     ;13455          MOV WR[0] TO LS[EPROR.MASK]     ;ERROR MASK TO CHECK NXM AND UB ADAP
                    ;13456                                          ; REG BITS OF CSR 1 ONLY
U 0F7A, B670,15     ;13457          MOV LS[OTHER.DATA] TO WR[0]     ;SET UP TO ENABLE PRINTOUT UNDER 'OTHER'
U 0F7B, 3E80,15     ;13458          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT IN CONTROL WORD
U 0F7C, 6588,15     ;13459          CLR LS[ADDRESS.DATA]            ;CLEAR ADDRESS TO PRINT
                    ;13460  LOOP.T1A.1:
U 0F7D, 2F82,95     ;13461          CLR WR[1]                       ;EXPECTED DATA
U 0F7E, 999C,75     ;13462          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                    ;13463                                          ;SET UP TO WRITE ADDRESS 0
U 0F7F, B29C,15     ;13464          WRITE.MEM LS[ZERO]              ;WRITE TO ADDRESS SPECIFIED
U 0F80, 9D45,75     ;13465          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;13466                                          ;SET UP TO READ CSR 1
U 0F81, 3022,15     ;13467          MOV MEM.DATA TO WR[0]           ;GET DATA FROM CSR 1
U 0F82, 0869,3C     ;13468          JSR [CHECK.RESULT]              ;CHECK CSR 1 FOR NXM AND UB ADAP REG CLEAR
U 0F83, 88F7,D4     ;13469          JMP [LOOP.T1A.1]                ;ERROR LOOP IF ENABLED
                    ;13470
U 0F84, FF82,15     ;13471          INC LS[ERROR.NUMBER]            ;ERROR 2
U 0F85, 3738,15     ;13472          MOV LS[#540000] TO WR[0]        ;HIGHEST POSSIBLE MEMORY ADDRESS + 1
U 0F86, BE12,15     ;13473          MOV WR[0] TO LS[T9]              ;STORE ADDRESS IN LS
                    ;13474  REPEAT.T1A.2:
U 0F87, 3612,15     ;13475          MOV LS[T9] TO WR[0]              ;ADDRESS STORED IN WR
U 0F88, BE88,15     ;13476          MOV WR[0] TO LS[ADDRESS.DATA]    ;GET ADDRESS TO PRINT IF ERROR
```

```
                    ;13477  LOOP.T1A.2:
U 0F89, B660,95     ;13478          MOV LS[NXM] TO WR[1]             ;EXPECTED DATA (NXM SET)
U 0F8A, 9912,75     ;13479          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                    ;13480                                           ;SET UP TO WRITE A NON-EXISTANT ADDRESS
U 0F8B, B29C,15     ;13481          WRITE.MEM LS[ZERO]               ;WRITE TO ADDRESS SPECIFIED
U 0F8C, 9D45,75     ;13482          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;13483                                           ;SET UP TO READ CSR 1
U 0F8D, 3022,15     ;13484          MOV MEM.DATA TO WR[0]            ;GET CONTENTS OF CSR 1
U 0F8E, 0869,3C     ;13485          JSR [CHECK.RESULT]               ;CHECK CSR 1 FOR NXM SET, UB ADAP REG
                    ;13486                                           ; CLEAR
U 0F8F, 08F8,94     ;13487          JMP [LOOP.T1A.2]                 ;ERROR LOOP IF ENABLED
                    ;13488
U 0F90, 3612,15     ;13489          MOV LS[T9] TO WR[0]              ;GET OLD ADDRESS
U 0F91, 4064,15     ;13490          ADD LS[#40000] TO WR[0]          ;INCREMENT BY 40000(H)
U 0F92, BE88,15     ;13491          MOV WR[0] TO LS[ADDRESS.DATA]    ;STORE IN LS FOR PRINTOUT IF ENABLED
U 0F93, BE12,15     ;13492          MOV WR[0] TO LS[T9]              ;STORE NEW ADDRESS IN LS
U 0F94, B73A,15     ;13493          MOV LS[#F00000] TO WR[0]         ;GET VALUE FROM LS TO COMPARE
                    ;13494          CMP WR[0] WITH LS[T9],           ;ARE WE AT ADDRESS F00000(H) YET?
U 0F95, 4F12,35     ;13495           DT(LONG)&SET.ALU.CC             ; SET CONDITION CODES
U 0F96, 88F8,71     ;13496          JMP.IF[NEQ] TO [REPEAT.T1A.2]    ;REPEAT IF NOT AT ADDRESS F00000(H)
                    ;13497
U 0F97, FF82,15     ;13498          INC LS[ERROR.NUMBER]             ;ERROR 3
                    ;13499  REPEAT.T1A.3:
U 0F98, 3612,15     ;13500          MOV LS[T9] TO WR[0]              ;GET ADDRESS FOR PRINTOUT
U 0F99, BE88,15     ;13501          MOV WR[0] TO LS[ADDRESS.DATA]    ;SAVE ADDRESS FOR PRINTOUT IF ERROR
                    ;13502  LOOP.T1A.3:
U 0F9A, 3664,95     ;13503          MOV LS[ADP.REG] TO WR[1]         ;EXPECTED DATA (UB ADAP REG SET)
U 0F9B, 9912,75     ;13504          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                    ;13505                                           ;SET UP TO WRITE A NON-EXISTANT ADDRESS
U 0F9C, B29C,15     ;13506          WRITE.MEM LS[ZERO]               ;WRITE TO ADDRESS SPECIFIED
U 0F9D, 9D45,75     ;13507          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;13508                                           ;SET UP TO READ CSR 1
U 0F9E, 3022,15     ;13509          MOV MEM.DATA TO WR[0]            ;GET CONTENTS OF CSR 1
U 0F9F, 0869,3C     ;13510          JSR [CHECK.RESULT]               ;CHECK CSR 1 FOR NXM CLEAR, UB ADAP REG
                    ;13511                                           ; SET
U 0FA0, 88F9,A4     ;13512          JMP [LOOP.T1A.3]                 ;ERROR LOOP IF ENABLED
                    ;13513
U 0FA1, 3612,15     ;13514          MOV LS[T9] TO WR[0]              ;GET OLD ADDRESS
U 0FA2, 4064,15     ;13515          ADD LS[#40000] TO WR[0]          ;INCREMENT BY 40000(H)
U 0FA3, BE12,15     ;13516          MOV WR[0] TO LS[T9]              ;STORE NEW ADDRESS IN LS
U 0FA4, B73C,15     ;13517          MOV LS[#FC0000] TO WR[0]         ;GET VALUE FROM LS TO COMPARE
                    ;13518          CMP WR[0] WITH LS[T9],           ;ARE WE AT ADDRESS FC0000(H) YET?
U 0FA5, 4F12,35     ;13519           DT(LONG)&SET.ALU.CC             ; SET CONDITION CODES
U 0FA6, 08F9,81     ;13520          JMP.IF[NEQ] TO [REPEAT.T1A.3]    ;REPEAT IF NOT AT ADDRESS FC0000(H)
                    ;13521
U 0FA7, FF82,15     ;13522          INC LS[ERROR.NUMBER]             ;ERROR 4
U 0FA8, DF2A,15     ;13523          MCOM LS[#FF000000] TO WR[0]      ;PUT ADDRESS FFFFFF(H) IN WR 0
U 0FA9, BE12,15     ;13524          MOV WR[0] TO LS[T9]              ;PUT IN LS FOR MEM.REQ
U 0FAA, BE88,15     ;13525          MOV WR[0] TO LS[ADDRESS.DATA]    ;PUT IN LS FOR ERROR ADDRESS PRINTOUT
                    ;13526  LOOP.T1A.4:
U 0FAB, 2F82,95     ;13527          CLR WR[1]                        ;EXPECTED DATA
U 0FAC, 9912,75     ;13528          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                    ;13529                                           ;SET UP TO WRITE A NON-EXISTANT ADDRESS
U 0FAD, B29C,15     ;13530          WRITE.MEM LS[ZERO]               ;WRITE TO ADDRESS SPECIFIED
U 0FAE, 9D45,75     ;13531          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
```

```
                 ;13532                                          ;SET UP TO READ CSR 1
U 0FAF, 3022,15  ;13533          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 0FB0, 0869,3C  ;13534          JSR [CHECK.RESULT]              ;CHECK CSR 1 FOR NXM CLEAR, UB ADAP REG
                 ;13535                                          ; CLEAR
U 0FB1, 08FA,B4  ;13536          JMP [LOOP.T1A.4]                ;ERROR LOOP IF ENABLED
                 ;13537
                 ;13538  END.T1A:
```

```
;13539  .PAGE   "  TEST 1B ILL UB OPER (CSR bit 20) Test (M8391 MCT or M8390 CPU Module)"
;13540  ;++
;13541  ;
;13542  ; Test Description:
;13543  ;
;13544  ;       This test checks bit 20 of CSR 1.  This is the illegal UB operation bit
;13545  ;       which will be set on an unaligned word write or any longword write to
;13546  ;       a UNIBUS device.  The device address used is all 1's or all 1's except
;13547  ;       bit 0.  This will always go to a non-existant device.
;13548  ;         The data path is as follows:  a MEM.REQ is issued with MF=WRITE.P and
;13549  ;       DT=longword.  The address used will be all ones except for bit 0.  The
;13550  ;       signal OP ERR L from the DIR AND MDR CONTROL PAL will be asserted and
;13551  ;       clocked into the CSR by the micro-code.  A WRITE.MEM LS will be issued
;13552  ;       to complete the write cycle and a read CSR 1 will be performed.  The
;13553  ;       ILL UB OPER bit (bit 20) will be checked for a 1.  The other tests will
;13554  ;       check a MEM.REQ with DT=WORD and with DT=BYTE with bit 0 both a 1 and a
;13555  ;       0.  The ILL UB OPER bit will be checked for the correct value.  Only
;13556  ;       bit 20 will be checked in this test. The other bits will be masked out.
;13557  ;         This test will also check that an unaligned word write to the array
;13558  ;       in compatability mode produces an ILL UB OPER error.
;13559  ;
;13560  ; Assumptions:
;13561  ;
;13562  ;       It is assumed that all previous tests have run successfully.
;13563  ;         This test uses some signals which originate on the CPU module (DATA
;13564  ;       TYPE 0, DATA TYPE 1, and COMPAT MODE).  These signals have been check-
;13565  ;       ed only on the CPU module itself, not coming out of the module.  So
;13566  ;       there is a slight chance that the CPU module could cause a failure.
;13567  ;       This is listed in the DEBUG section.
;13568  ;
;13569  ; Test Steps:
;13570  ;
;13571  ;           1)  Set up error number and module number in LS (for error printouts)
;13572  ;               and clear previous error number in LS.
;13573  ;           2)  Set up error mask to check bit 20 (ILL UB OPER) only.  Clear COM-
;13574  ;               PAT mode bit (bit 31) in PSL.
;13575  ;           3)  Execute MEM.REQ with MF=WRITE.P and DT=longword.  Use an LS loc-
;13576  ;               ation containing all 1's except for bit 0 as the address.
;13577  ;           4)  Execute WRITE.MEM LS with data of zeros.  Then read the CSR and
;13578  ;               check ILL UB OPER for a 1.
;13579  ;           5)  Repeat steps 3 and 4 with DT=word (step 3) and check ILL UB OPER
;13580  ;               for a 0 (step 4).  This is a legal UNIBUS acce s.
;13581  ;           6)  Repeat steps 3 and 4 with DT= word and address-all 1's.  This is
;13582  ;               an illegal UNIBUS access (not word aligned).
;13583  ;           7)  Repeat steps 3 and 4 with DT= byte and address=all 1's.  This is
;13584  ;               a legal UNIBUS access (byte data needn't be aligned).
;13585  ;           8)  Repeat steps 3 and 4 with DT= byte and address containong all 1's
;13586  ;               except for bit 0.  This is a legal UNIBUS access (aligned byte).
;13587  ;           9)  Set the COMPAT mode bit (bit 31) in the PSL.
;13588  ;           10) Execute MEM.REQ with MF=WRITE.P and DT=word.  Use an LS location
;13589  ;               containing all 0's except for bit 0 as the address.
;13590  ;           11) Execute WRITE.MEM LS with data of zeros.  Then read the CSR and
;13591  ;               check ILL UB OPER for a 1 (unaligned word write in COMPAT mode)
;13592  ;           12) Clear the COMPAT moce bit in the PSL and repeat steps 9 and 10.
;13593  ;               Check ILL UB OPER for a 0 (unaligned word write is legal in native
```

```
;13594  ;           mode).
;13595  ;
;13596  ; Errors:
;13597  ;
;13598  ;       NOTE:  expected and received data is CSR 1 bit 20 (OP ERR).
;13599  ;
;13600  ;       Error 1 - ILL UB OPER in CSR failed to set on longword write to UNIBUS.
;13601  ;       Error 2 - ILL UB OPER in CSR failed to clear on aligned word write to
;13602  ;                 UNIBUS.
;13603  ;       Error 3 - ILL UB OPER in CSR failed to set on unaligned word write to
;13604  ;                 UNIBUS.
;13605  ;       Error 4 - ILL UB OPER in CSR failed to clear on byte write to UNIBUS
;13606  ;                 (odd address).
;13607  ;       Error 5 - ILL UB OPER in CSR failed to clear on byte write to UNIBUS
;13608  ;                 (even address).
;13609  ;       Error 6 - ILL UB OPER in CSR failed to set on unaligned word write to
;13610  ;                 array in compatability mode.
;13611  ;       Error 7 - ILL UB OPER in CSR failed to clear on unaligned word write to
;13612  ;                 array in native mode.
;13613  ;
;13614  ; Debug:
;13615  ;
;13616  ;       Two methods of debug will be explained.  If the MCT module is in a
;13617  ;       test station then the MCT can be single stepped making debug easier
;13618  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;13619  ;       method will be explained also.  When single stepping the memory con-
;13620  ;       troller will help in debug, this section will explicitly suggest it.
;13621  ;
;13622  ;       Error 1 - This error indicates a problem with the DECODER A PAL, the
;13623  ;       DATA ROTATOR CONTROL PAL, the MDR AND DIR CONTROL PAL, or the CSR 1A
;13624  ;       PAL.  Single step the CPU to the MEM.REQ with MF=WRITE.P.  The memory
;13625  ;       micro-code will loop waiting for CPU DATA REQ.  Check the output signal
;13626  ;       UB PH ADDR SEL L of the DECODER A PAL for a low.  If this is incorrect,
;13627  ;       check the inputs for highs on PA 23 H through PA 18 H.  If these inputs
;13628  ;       are OK, the PAL is bad.
;13629  ;         If UB PH ADDR SEL L is low, check it at the input to the MDR AND DIR
;13630  ;       CONTROL PAL.  The output signal OP ERR L from this PAL should be low.
;13631  ;       If not, check for a low on the input RS1 L as well as a low on UB PH
;13632  ;       ADDR SEL L.  If these inputs are OK, the PAL is bad.
;13633  ;         If RS1 L is high, check it at the output of the DATA ROTATOR CONTROL
;13634  ;       PAL.  If incorrect there, check the inputs for a high on CPH/UBL and
;13635  ;       a high on L DT1 H.  L DT1 H can be traced back through the latch to
;13636  ;       the CPU module if it is incorrect.  If the inputs are correct, the PAL
;13637  ;       is probably bad.
;13638  ;         If OP ERR L is low out of the MDR AND DIR CONTROL PAL, check it at
;13639  ;       input to the CSR 1A PAL.  If OK there, the CSR 1A PAL is probably bad.
;13640  ;
;13641  ;       Error 2 - This error indicates a problem with the DATA ROTATOR CONTROL
;13642  ;       PAL, the MDR AND DIR CONTROL PAL, or the CSR 1A PAL.  Single step the
;13643  ;       CPU to the MEM.REQ with MF=WRITE.P.  The memory micro-code will loop
;13644  ;       waiting for CPU DATA REQ.  Check the output signal OP ERR L from the
;13645  ;       MDR AND DIR CONTROL PAL for a high.  If this is incorrect, check the
;13646  ;       inputs for a high on RS1 L and A0 L.  If these are OK, then the MDR AND
;13647  ;       DIR CONTROL PAL is probably bad.
;13648  ;         If either RS1 L or A0 L is low, check them at the output of the DATA
```

```
;13649  ;       ROTATOR CONTROL PAL.  If incorrect there, check the inputs for a low
;13650  ;       on L DT1 H and a low on LVA 00 H.  L DT1 H can be traced back through
;13651  ;       the 8 bit latch that produces it if incorrect.
;13652  ;         If OP ERR L out of the MDR AND DIR CONTROL PAL is high, check it at
;13653  ;       the input to the CSR 1A PAL.  If correct there, the CSR 1A PAL is prob-
;13654  ;       ably bad.
;13655  ;
;13656  ;       Error 3 - This error indicates a problem with either the DATA ROTATOR
;13657  ;       CONTROL PAL or the MDR AND DIR CONTROL PAL.  Single step the CPU to the
;13658  ;       MEM.REQ with MF=WRITE.P.  The memory micro-code will loop waiting for
;13659  ;       CPU DATA REQ.  Check the output signal OP ERR L from the MDR AND DIR
;13660  ;       CONTROL PAL for a low.  If this is incorrect, check the inputs for a
;13661  ;       low on RS0 L and a low on A0 L.  If these are OK, then the MDR AND DIR
;13662  ;       CONTROL PAL is probably bad.
;13663  ;         If either RS0 L or A0 L is high, check them at the output of the DATA
;13664  ;       ROTATOR CONTROL PAL.  If incorrect there, check the inputs for highs on
;13665  ;       LVA0 H and L DT0 H.  If these are OK, the DATA ROTATOR CONTROL PAL is
;13666  ;       bad.  L DT0 H can be traced back through the 8 bit latch that outputs
;13667  ;       it if incorrect.
;13668  ;
;13669  ;       Error 4 - This error indicates a problem with either the DATA ROTATOR
;13670  ;       CONTROL PAL or the MDR AND DIR CONTROL PAL.  Single step the CPU to the
;13671  ;       MEM.REQ with MF=WRITE.P.  The memory micro-code will loop waiting for
;13672  ;       CPU DATA REQ.  Check the output signal OP ERR L from the MDR AND DIR
;13673  ;       CONTROL PAL for a high.  If this is incorrect, check the inputs for a
;13674  ;       high on RS1 L and a high on RS0 L.  If these are OK, then the MDR AND
;13675  ;       DIR CONTROL PAL is probably bad.
;13676  ;         If either RS1 L or RS0 L is low, check them at the output of the DATA
;13677  ;       ROTATOR CONTROL PAL.  If incorrect there, check the inputs for a low on
;13678  ;       L DT0 H and L DT1 H.  If these are correct, the PAL is probably bad.
;13679  ;       L DT0 H can be traced back to the 8 bit latch that outputs it if incor-
;13680  ;       rect.
;13681  ;
;13682  ;       Error 5 - Same as error 4.  If this is the first error, suspect the MDR
;13683  ;       AND DIR CONTROL PAL.
;13684  ;
;13685  ;       Error 6 - This error indicates a problem with the MDR AND DIR CONTROL
;13686  ;       PAL or its COMPAT MODE H input.  Single step the CPU to the MEM.REQ
;13687  ;       with MF=WRITE.P.  The memory micro-code will loop waiting for CPU DATA
;13688  ;       REQ.  Check the output signal OP ERR L from the MDR AND DIR CONTROL PAL
;13689  ;       for a low.  If this is incorrect, check the input for a high on COMPAT
;13690  ;       MODE H.  If this is correct, then the MDR AND DIR CONTROL PAL is prob-
;13691  ;       ably bad (the other inputs were tested previously).
;13692  ;         If the signal COMPAT MODE H is low, it can be traced back to the CPU
;13693  ;       module.  It has been previously tested on the CPU module, but not com-
;13694  ;       ing out of the module.
;13695  ;
;13696  ;       Error 7 - This error indicates a problem with the MDR AND DIR CONTROL
;13697  ;       PAL, its inputs, or the DECODER A PAL.  Single step the CPU to the
;13698  ;       MEM.REQ with MF=WRITE.P.  The memory micro-code will loop waiting for
;13699  ;       CPU DATA REQ.  Check the output signal OP ERR L from the MDR AND DIR
;13700  ;       CONTROL PAL for a high.  If this is incorrect, check the inputs for a
;13701  ;       low on COMPAT MODE H and a high on UB PH ADDR SEL L.  If these are cor-
;13702  ;       rect, then the MDR AND DIR CONTROL PAL is probably bad.
;13703  ;         If the signal COMPAT MODE H is high, it can be traced back to the CPU
```

```
                    ;13704  ;       module.  It has been previously tested on the CPU module, but not com-
                    ;13705  ;       ing out of the module.
                    ;13706  ;         If UB PH ADDR SEL L is low, check it coming out of the DECODER A PAL.
                    ;13707  ;       If incorrect there, check the inputs for a low on any of the PA 23 H
                    ;13708  ;       through PA 18 H lines.  The PAL is bad if these inputs are correct (any
                    ;13709  ;       low on PA 23 through PA 18 should drive UB PH ADDR SEL L high).
                    ;13710  ;
                    ;13711  ;--
                    ;13712
                    ;13713  T.1B:
                    ;13714
U 0FB2, B65E,15     ;13715          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;13716                                          ; STATUS WORD
U 0FB3, 3E80,15     ;13717          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;13718                                          ; BIT 15 INDICATES START OF TEST TO
                    ;13719                                          ; CONSOLE
U 0FB4, 10E0,15     ;13720          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;13721  WAIT.T1B.0:
U 0FB5, 08FB,54     ;13722          JMP [WAIT.T1B.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;13723                                          ; RESPOND
                    ;13724
U 0FB6, 0A1A,AC     ;13725          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;13726
U 0FB7, 4742,15     ;13727          BIS LS[CPU] TO WR[0]            ;ALSO SET BIT 1 FOR CPU MODULE
U 0FB8, 3E8C,15     ;13728          MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                    ;13729                                          ; MCT AND CPU BOARD
U 0FB9, DF68,15     ;13730          MCOM LS[OP.ERR] TO WR[0]        ;CLEAR BIT 20 IN WR 0
U 0FBA, 3E8A,15     ;13731          MOV WR[0] TO LS[ERROR.MASK]     ;SET UP ERROR MASK TO CHECK BIT 20 ONLY
U 0FBB, 2F80,15     ;13732          CLR WR[0]                       ;USED TO LOAD PSL
U 0FBC, BFFE,15     ;13733          MOV WR[0] TO LS[PSL.HW]          ;CLEAR PSL TO ASSURE COMPAT MODE CLEAR
U 0FBD, DF40,15     ;13734          MCOM LS[BIT0] TO WR[0]          ;GET VALUE OF FFFFFFFE(H) IN WR
U 0FBE, BE12,15     ;13735          MOV WR[0] TO LS[T9]              ;LS 9 CONTAINS ADDRESS
                    ;13736  LOOP.T1B.1:
U 0FBF, 3668,95     ;13737          MOV LS[OP.ERR] TO WR[1]          ;EXPECTED DATA (ILL UB OP ERR)
U 0FC0, 9912,75     ;13738          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                    ;13739                                          ;SET UP TO WRITE LONGWORD TO UNIBUS
U 0FC1, B29C,15     ;13740          WRITE.MEM LS[ZERO]              ;WRITE ILLEGAL LONGWORD TO UNIBUS
U 0FC2, 9D45,75     ;13741          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;13742                                          ;SET UP TO READ CSR 1
U 0FC3, 3022,15     ;13743          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 0FC4, 0869,3C     ;13744          JSR [CHECK.RESULT]              ;CHECK ILL UB OP ERR FOR A 1
U 0FC5, 08FB,F4     ;13745          JMP [LOOP.T1B.1]                ;ERROR LOOP IF ENABLED
                    ;13746
U 0FC6, FF82,15     ;13747          INC LS[ERROR.NUMBER]            ;ERROR 2
                    ;13748  LOOP.T1B.2:
U 0FC7, 2F82,95     ;13749          CLR WR[1]                       ;EXPECTED DATA (ILL UB OP CLEAR)
U 0FC8, 1912,35     ;13750          MEM.REQ[WRITE.P] ADRS[T9] DT[WORD]
                    ;13751                                          ;SET UP TO WRITE WORD TO UNIBUS
U 0FC9, B29C,15     ;13752          WRITE.MEM LS[ZERO]              ;WRITE LEGAL WORD TO UNIBUS
U 0FCA, 9D45,75     ;13753          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;13754                                          ;SET UP TO READ CSR 1
U 0FCB, 3022,15     ;13755          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 0FCC, 0869,3C     ;13756          JSR [CHECK.RESULT]              ;CHECK ILL UB OP ERR FOR A 0
U 0FCD, 08FC,74     ;13757          JMP [LOOP.T1B.2]                ;ERROR LOOP IF ENABLED
                    ;13758
```

```
U 0FCE, FF82,15 ;13759          INC LS[ERROR.NUMBER]             ;ERROR 3
                ;13760  LOOP.T1B.3:
U 0FCF, 3668,95 ;13761          MOV LS[OP.ERR] TO WR[1]          ;EXPECTED DATA (ILL UB OP ERR)
U 0FD0, 999E,35 ;13762          MEM.REQ[WRITE.P] ADRS[ONES] DT[WORD]
                ;13763                                           ;SET UP TO WRITE WORD TO UNIBUS
U 0FD1, B29C,15 ;13764          WRITE.MEM LS[ZERO]               ;WRITE ILLEGAL WORD TO UNIBUS (NOT
                ;13765                                           ; WORD ALIGNED
U 0FD2, 9D45,75 ;13766          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;13767                                           ;SET UP TO READ CSR 1
U 0FD3, 3022,15 ;13768          MOV MEM.DATA TO WR[0]            ;GET CONTENTS OF CSR 1
U 0FD4, 0869,3C ;13769          JSR [CHECK.RESULT]               ;CHECK ILL UB OP ERR FOR A 1
U 0FD5, 88FC,F4 ;13770          JMP [LOOP.T1B.3]                 ;ERROR LOOP IF ENABLED
                ;13771
U 0FD6, FF82,15 ;13772          INC LS[ERROR.NUMBER]             ;ERROR 4
                ;13773  LOOP.T1B.4:
U 0FD7, 2F82,95 ;13774          CLR WR[1]                        ;EXPECTED DATA (ILL UB OP CLEAR)
U 0FD8, 199E,15 ;13775          MEM.REQ[WRITE.P] ADRS[ONES] DT[BYTE]
                ;13776                                           ;SET UP TO WRITE BYTE TO UNIBUS
U 0FD9, B29C,15 ;13777          WRITE.MEM LS[ZERO]               ;WRITE LEGAL BYTE TO UNIBUS (NEED NOT
                ;13778                                           ; BE WORD ALIGNED)
U 0FDA, 9D45,75 ;13779          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;13780                                           ;SET UP TO READ CSR 1
U 0FDB, 3022,15 ;13781          MOV MEM.DATA TO WR[0]            ;GET CONTENTS OF CSR 1
U 0FDC, 0869,3C ;13782          JSR [CHECK.RESULT]               ;CHECK ILL UB OP ERR FOR A 0
U 0FDD, 88FD,74 ;13783          JMP [LOOP.T1B.4]                 ;ERROR LOOP IF ENABLED
                ;13784
U 0FDE, FF82,15 ;13785          INC LS[ERROR.NUMBER]             ;ERROR 5
                ;13786
                ;13787  LOOP.T1B.5:
U 0FDF, 2F82,95 ;13788          CLR WR[1]                        ;EXPECTED DATA (ILL UB OP CLEAR)
U 0FE0, 9912,15 ;13789          MEM.REQ[WRITE.P] ADRS[T9] DT[BYTE]
                ;13790                                           ;SET UP TO WRITE BYTE TO UNIBUS
U 0FE1, B29C,15 ;13791          WRITE.MEM LS[ZERO]               ;WRITE LEGAL BYTE TO UNIBUS (EVEN
                ;13792                                           ; ADDRESS)
U 0FE2, 9D45,75 ;13793          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;13794                                           ;SET UP TO READ CSR 1
U 0FE3, 3022,15 ;13795          MOV MEM.DATA TO WR[0]            ;GET CONTENTS OF CSR 1
U 0FE4, 0869,3C ;13796          JSR [CHECK.RESULT]               ;CHECK ILL UB OP ERR FOR A 0
U 0FE5, 08FD,F4 ;13797          JMP [LOOP.T1B.5]                 ;ERROR LOOP IF ENABLED
                ;13798
U 0FE6, FF82,15 ;13799          INC LS[ERROR.NUMBER]             ;ERROR 6
                ;13800  LOOP.T1B.6:
U 0FE7, 3668,95 ;13801          MOV LS[OP.ERR] TO WR[1]          ;EXPECTED DATA (ILL UB OP ERR SET)
U 0FE8, 367E,15 ;13802          MOV LS[BIT31] TO WR[0]           ;SET BIT 31
U 0FE9, BFFE,15 ;13803          MOV WR[0] TO LS[PSL.HW]          ;SET COMPAT MODE
U 0FEA, 9940,35 ;13804          MEM.REQ[WRITE.P] ADRS[#1] DT[WORD]
                ;13805                                           ;SET UP TO WRITE WORD TO ODD MEMORY
                ;13806                                           ; ADDRESS
U 0FEB, B29C,15 ;13807          WRITE.MEM LS[ZERO]               ;WRITE ILLEGAL WORD TO MEMORY (ODD
                ;13808                                           ; ADDRESS IN COMPATABILITY MODE)
U 0FEC, 2F80,15 ;13809          CLR WR[0]                        ;CLEAR WR
U 0FED, BFFE,15 ;13810          MOV WR[0] TO LS[PSL.HW]          ;CLEAR COMPAT MODE IN PSL
U 0FEE, 9D45,75 ;13811          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;13812                                           ;SET UP TO READ CSR 1
U 0FEF, 3022,15 ;13813          MOV MEM.DATA TO WR[0]            ;GET CONTENTS OF CSR 1
```

```
U 0FF0, 0869,3C ;13814          JSR [CHECK.RESULT]              ;CHECK ILL UB OP ERR FOR A 1
U 0FF1, 88FE,74 ;13815          JMP [LOOP.T1B.6]                ;ERROR LOOP IF ENABLED
                ;13816
U 0FF2, FF82,15 ;13817          INC LS[ERROR.NUMBER]            ;ERROR 7
                ;13818  LOOP.T1B.7:
U 0FF3, 2F82,95 ;13819          CLR WR[1]                       ;EXPECTED DATA (ILL UP OP ERR CLEAR)
U 0FF4, 9940,35 ;13820          MEM.REQ[WRITE.P] ADRS[#1] DT[WORD]
                ;13821                                          ;SET UP TO WRITE WORD TO ODD MEMORY
                ;13822                                          ; ADDRESS
U 0FF5, B29C,15 ;13823          WRITE.MEM LS[ZERO]              ;WRITE LEGAL WORD TO MEMORY (ODD
                ;13824                                          ; ADDRESS IN NATIVE MODE)
U 0FF6, 9D45,75 ;13825          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;13826                                          ;SET UP TO READ CSR 1
U 0FF7, 3022,15 ;13827          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 0FF8, 0869,3C ;13828          JSR [CHECK.RESULT]              ;CHECK ILL UB OP ERR FOR A 0
U 0FF9, 88FF,34 ;13829          JMP [LOOP.T1B.7]                ;ERROR LOOP IF ENABLED
                ;13830
                ;13831  END.T1B:
U 0FFA, 0900,04 ;13832          JMP [T.1C]                      ;NEXT TEST STARTS AT 1000 TO AVOID
                ;13833                                          ; VALIDITY FAILURES
```

```
;13834  .PAGE   "  TEST 1C Write Across Page Error Bit (Bit 19) (M8391  MCT Module)"
;13835  ;++
;13836  ;
;13837  ; Test Description:
;13838  ;
;13839  ;       This test checks the WR ACROSS PG ERR bit of CSR 1 (bit 19).  This bit
;13840  ;       is set when a write to a certain virtual address would require two mem-
;13841  ;       ory cycles and would cross either a page boundary (virtual address bits
;13842  ;       2 through 8 all set) or a system address boundary (bits 2 through 29
;13843  ;       all set).  The TEST.V.WCHK flow will be used to check this signal.  The
;13844  ;       data path is as follows:  a MEM.REQ is issued with MF=TEST.V.WCHK and
;13845  ;       DT=LONGWORD.  The TEST.V.WCHK micro-flow issues a ROT CLK which clocks
;13846  ;       the DATA ROTATOR CONTROL PAL.  This PAL outputs the signal 2 MEM CYCLES
;13847  ;       L, which is asserted if the low two bits (LVA 00 and LVA 01) are not 0
;13848  ;       on a longword write or if the low two bits are both high on a word
;13849  ;       write.  2 MEM CYCLES L then goes to the CSR 1B PAL.  The input PAGE
;13850  ;       BOUNDARY H to the CSR 1B PAL comes from the VAR.  If VAR 02 through 08
;13851  ;       are set, PAGE BOUNDARY H goes H, and will cause a WR ACROSS PG ERR if
;13852  ;       2 MEM CYCLES L is asserted.  After the MEM.REQ a MOV MEM.DATA TO WR is
;13853  ;       issued to complete the TEST cycle and the CSR is read to check the WR
;13854  ;       ACROSS PG ERR bit.
;13855  ;         A WR ACROSS PG ERR will also come up if the write would cause address
;13856  ;       bit 30 or 31 to change.  This occurs if 2 MEM CYCLES L is asserted and
;13857  ;       SYS ADDR VIOL H is asserted.  SYS ADDR VIOL H comes from the VAR logic
;13858  ;       and is asserted if bits 02 through 29 are all set.
;13859  ;         This test will check the bit on a byte, word, and longword access
;13860  ;       with various combinations of LVA 00 and LVA 01 to assure that the bit
;13861  ;       works correctly.
;13862  ;
;13863  ; Assumptions:
;13864  ;
;13865  ;       It is assumed that all previous tests have run successfully.
;13866  ;
;13867  ; Test Steps:
;13868  ;
;13869  ;       1)  Set up error number and module number in LS (for error printouts)
;13870  ;           and clear previous error number in LS.
;13871  ;       2)  Set up error mask to check bit 19 (WR ACROSS PG ERR) only and write
;13872  ;           CSR 1 to set MME.
;13873  ;       3)  Execute MEM.REQ with MF=TEST.V.WCHK and DT=longword.  Use an LS
;13874  ;           location containing 1's in bits 2 through 8 (produces PAGE BOUN-
;13875  ;           DARY H) and zeros in bits 0 and 1.
;13876  ;       4)  Execute MOV MEM.DATA TO WR[0] to complete the TEST.V.WCHK cycle
;13877  ;           but don't check this data.
;13878  ;       5)  Read the CSR and check WR ACROSS PG ERR for a low.
;13879  ;       6)  Repeat steps 3 and 4 except with a 01(B) in bits 1 and 0.  Read the
;13880  ;           CSR and check WR ACROSS PG ERR for a high (set).
;13881  ;       7)  Repeat step 6 with 10(B) in bits 1 and 0 and with 11(B) in bits 1
;13882  ;           and 0.
;13883  ;       8)  Repeat steps 3 through 5 with DT=word and with 10(B) in bits 1 and
;13884  ;           0.
;13885  ;       9)  Repeat steps 3 through 5 with DT=byte and with 11(B) in bits 1 and
;13886  ;           0.
;13887  ;       10) Execute MEM.REQ with MF=TEST.V.WCHK and DT=longword.  Use an LS
;13888  ;           location containing ones in bits 02 through 08 except for a shift-
```

```
;13889  ;           ing 0 in one bit.  Bits 1 and 0 contain a 11(B) to assert 2 MEM
;13890  ;           CYCLES.
;13891  ;       11) Execute MOV MEM.DATA TO WR[0] to complete the TEST.V.WCHK cycle
;13892  ;           but don't check this data.
;13893  ;       12) Read the CSR and check WR ACROSS PG ERR for a low (clear).  Repeat
;13894  ;           steps 10 through 12 until a zero has been shifted across each bit
;13895  ;           of VAR bits 02 through 08.
;13896  ;       13) Write a 1 in CSR 1 in the TB PAR DIAG bit to force a parity error
;13897  ;           on any access attempted through the TB.
;13898  ;       14) Execute MEM.REQ with MF=WRITE.V.NOCHK and DT=longword.  Use an LS
;13899  ;           location containing 1's in bits 0 through 8.
;13900  ;       15) Execute WRITE.MEM LS to complete the WRITE.V.NOCHK cycle.
;13901  ;       16) Read the CSR and check WR ACROSS PG ERR for a low (clear).  (NOCHK
;13902  ;           will not cause a WR ACROSS PG ERR).
;13903  ;       17) Repeat steps 14 and 15 with 1's in bits 0 through 29 in step 14
;13904  ;           (this will cause SYS ADDR VIOL H to be asserted).  Check WR ACROSS
;13905  ;           PG ERR for a high (set).
;13906  ;
;13907  ; Errors:
;13908  ;
;13909  ;       NOTE:  Expected and received data is CSR 1 bit 19 (WR XPG ERR).  OTHER
;13910  ;              data contains address accessed.
;13911  ;
;13912  ;       Error 1 - WR ACROSS PG ERR set on aligned longword check after TEST.V
;13913  ;                 .WCHK command.
;13914  ;       Error 2 - WR ACROSS PG ERR clear on unaligned longword check after TEST
;13915  ;                 .V.WCHK command.
;13916  ;       Error 3 - WR ACROSS PG ERR set on aligned word check after TEST.V.WCHK
;13917  ;                 command.
;13918  ;       Error 4 - WR ACROSS PG ERR set on byte check after TEST.V.WCHK command.
;13919  ;       Error 5 - WR ACROSS PG ERR set on unaligned longword check without a
;13920  ;                 page boundary after TEST.V.WCHK command.
;13921  ;       Error 6 - WR ACROSS PG ERR set on unaligned longword check after WRITE
;13922  ;                 .V.NOCHK command.
;13923  ;       Error 7 - WR ACROSS PG ERR clear on unaligned longword check with SYS
;13924  ;                 ADDR VIOL asserted after TEST.V.NOCHK command.
;13925  ;
;13926  ; Debug:
;13927  ;
;13928  ;       Two methods of debug will be explained.  If the MCT module is in a
;13929  ;       test station then the MCT can be single stepped making debug easier
;13930  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;13931  ;       method will be explained also.  When single stepping the memory con-
;13932  ;       troller will help in debug, this section will explicitly suggest it.
;13933  ;
;13934  ;       Note: OTHER contains the address used with the TEST.V.WCHK command.
;13935  ;
;13936  ;       Error 1 - This error indicates a problem with the DATA ROTATOR CONTROL
;13937  ;       PAL or the CSR 1B PAL.  Single step the CPU to the MEM.REQ with MF=
;13938  ;       TEST.V.WCHK.  The memory u-code will loop waiting for CPU GRANT to go
;13939  ;       away (this would occur after the MOV MEM.DATA TO WR instruction was
;13940  ;       executed).  Check the signal 2 MEM CYCLES L into the CSR 1B PAL for a
;13941  ;       high.  If this is correct, the output BUS MC D19 H should be low.  If
;13942  ;       not, the CSR 2 PAL is probably bad.
;13943  ;         If 2 MEM CYCLES L is low at the CSR 1B PAL, check it at the output
```

```
;13944  ;       of the DATA ROTATOR CONTROL PAL.  If incorrect (low) there, check the
;13945  ;       inputs for lows on LVA 01 H and LVA 00 H.  If these are correct, then
;13946  ;       the DATA ROTATOR CONTROL PAL is probably bad.  If not, trace them back
;13947  ;       to the VAR logic.
;13948  ;
;13949  ;       Error 2 - This error indicates a problem with the DATA ROTATOR CONTROL
;13950  ;       PAL, the CSR 1B PAL or the VAR logic that produces PAGE BOUNDARY H.
;13951  ;       Single step the CPU to the MEM.REQ with MF=TEST.V.WCHK.  The memory
;13952  ;       u-code will loop waiting for CPU GRANT to go away (this would occur
;13953  ;       after the MOV MEM.DATA TO WR instruction was executed).  Check the
;13954  ;       output BUS MC D19 H from the CSR 1B PAL for a high.  If this is incor-
;13955  ;       rect, check the inputs for a low on 2 MEM CYCLES L and a high on PAGE
;13956  ;       BOUNDARY H.  Also check that RD CSR L is not floating (it will be high
;13957  ;       at this point but needs to have gone low during the TEST.V.WCHK flow).
;13958  ;       If these inputs are OK, the PAL is probably bad.  The signal RD CSR L
;13959  ;       can be checked for a low by single stepping the MCT (if MCT single step
;13960  ;       is available).  Single step the CPU to the instruction preceeding the
;13961  ;       MEM.REQ and enable MCT single step.  Then step the CPU to the MEM.REQ
;13962  ;       and step the MCT to the 2nd cycle following the dispatch address.  RD
;13963  ;       CSR L should be low along with a high on CSR ERR SUM CLK H.  If MCT
;13964  ;       single step is not available, the signal can be checked while looping
;13965  ;       on error.
;13966  ;         If 2 MEM CYCLES L is high, check it at the output of the DATA ROTATOR
;13967  ;       CONTROL PAL.  If incorrect (high) there, check the inputs for a high on
;13968  ;       LVA 00 H or LVA 01 H.  If either of these is high, the PAL is probably
;13969  ;       bad.  These inputs can be traced back to the VAR if incorrect.
;13970  ;         If PAGE BOUNDARY H is low at the CSR 1B PAL, check it at the output
;13971  ;       of the VAR.  If incorrect (low) there, suspect the VAR PAL itself.
;13972  ;
;13973  ;       Error 3 - This error most likely indicates a problem with the DATA
;13974  ;       ROTATOR CONTROL PAL itself.  The inputs have been checked in previous
;13975  ;       tests.
;13976  ;
;13977  ;       Error 4 - Same as error 3.
;13978  ;
;13979  ;       Error 5 - This error indicates a problem with the CSR 1B PAL or the VAR
;13980  ;       logic that produces PAGE BOUNDARY H and SYS ADDR VIOL H.  Single step
;13981  ;       the CPU to the MEM.REQ with MF=TEST.V.WCHK.  The memory u-code will
;13982  ;       loop waiting for CPU GRANT to go away (this would occur after the MOV
;13983  ;       MEM.DATA TO WR instruction was executed).  Check the inputs to the CSR
;13984  ;       1B PAL for lows on PAGE BOUNDARY H and SYS ADDR VIOL H.  If these in-
;13985  ;       puts are OK, then the CSR 1B PAL is probably bad.
;13986  ;         If PAGE BOUNDARY H is high, suspect the VAR PAL that produces it.
;13987  ;         If SYS ADDR VIOL H is high, check it at the VAR PAL that produces it.
;13988  ;       If high there, suspect that VAR PAL.
;13989  ;
;13990  ;       Error 6 - This error indicates a problem with the CSR 1B PAL or the in-
;13991  ;       put RD CSR L.  Single step the CPU to the MEM.REQ with MF=WRITE.V
;13992  ;       .NOCHK.  The memory u-code will loop waiting for CPU GRANT to go away
;13993  ;       (this would occur after the WRITE.MEM LS instruction was executed).
;13994  ;       Check the input RD CSR L for a high.  If this is OK, the CSR 1B PAL is
;13995  ;       probably bad.
;13996  ;
;13997  ;       Error 7 - This error indicates a problem with the CSR 1B PAL or the VAR
;13998  ;       logic that produces SYS ADDR VIOL H.  Single step the CPU to the MEM
```

```
                    ;13999  ;       .REQ with MF=WRITE.V.NOCHK.  The memory u-code will loop waiting for
                    ;14000  ;       CPU GRANT to go away (this would occur after the WRITE.MEM LS instruc-
                    ;14001  ;       tion was executed).  Check the inputs to the CSR 1B PAL for a high on
                    ;14002  ;       SYS ADDR VIOL H.  If this is OK, the CSR 1B PAL is probably bad.
                    ;14003  ;         If SYS ADDR VIOL H is low, check it at the output of the VAR PAL that
                    ;14004  ;       produces it.  The VAR PAL is probably bad if the output is low.
                    ;14005  ;
                    ;14006  ;--
                    ;14007
                    ;14008  1000:                                   ;START AT 1000 TO AVOID VALIDITY FAILURES
                    ;14009  T.1C:
                    ;14010
U 1000, B65E,15     ;14011          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;14012                                          ; STATUS WORD
U 1001, 3E80,15     ;14013          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;14014                                          ; BIT 15 INDICATES START OF TEST TO
                    ;14015                                          ; CONSOLE
U 1002, 10E0,15     ;14016          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;14017  WAIT.T1C.0:
U 1003, 0900,34     ;14018          JMP [WAIT.T1C.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;14019                                          ; RESPOND
                    ;14020
U 1004, 0A1A,AC     ;14021          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;14022
U 1005, 5F66,15     ;14023          MCOM LS[WR.ACROSS.PG] TO WR[0]  ;CLEAR BIT 19 IN WR
U 1006, 3E8A,15     ;14024          MOV WR[0] TO LS[ERROR.MASK]     ;SET UP ERROR MASK TO CHECK WRITE ACROSS
                    ;14025                                          ; PAGE ERROR BIT ONLY
                    ;14026
U 1007, 0A17,DC     ;14027      JSR [WRITE.CSR1.MME]                ;ENABLE MEMORY MANAGEMENT
U 1008, B670,15     ;14028      MOV LS[OTHER.DATA] TO WR[0]         ;SET BIT TO PRINT UNDER OTHER IN ERROR
                    ;14029                                          ; REPORT
U 1009, 3E80,15     ;14030      MOV WR[0] TO LS[CONTROL.STATUS]     ;WRITE IN CONTROL AND STATUS WORD
U 100A, B626,15     ;14031      MOV LS[#FF] TO WR[0]                ;SET BITS 0-7 IN WR
U 100B, 4750,15     ;14032      BIS LS[BIT8] TO WR[0]               ;SET BIT 8
U 100C, C5C0,15     ;14033      BIC LS[#3(H)] TO WR[0]              ;CLEAR BITS 0 AND 1
U 100D, BE12,15     ;14034      MOV WR[0] TO LS[T9]                 ;LOAD ADDRESS IN LS FOR MEM.REQ
U 100E, BE88,15     ;14035      MOV WR[0] TO LS[ADDRESS.DATA]       ;ADDRESS FOR OTHER DATA IN ERROR REPORT
U 100F, 2F87,95     ;14036      CLR WR[3]                           ;EXPECTED DATA (WR ACROSS PG ERR CLEAR)
U 1010, 0905,EC     ;14037      JSR [LOOP.T1C.1.2]                  ;DO TEST WITH ADDRESS OF 1FC
                    ;14038
U 1011, FF82,15     ;14039      INC LS[ERROR.NUMBER]                ;ERROR 2
U 1012, FF12,15     ;14040      INC LS[T9]                          ;ADDRESS NOW HAS 01(B) IN LOW 2 BITS
U 1013, FF88,15     ;14041      INC LS[ADDRESS.DATA]                ;ADDRESS TO PRINT UNDER OTHER
U 1014, 3667,95     ;14042      MOV LS[WR.ACROSS.PG] TO WR[3]       ;EXPECTED DATA (WR ACROSS PAGE ERR SET)
U 1015, 0905,EC     ;14043      JSR [LOOP.T1C.1.2]                  ;DO TEST WITH ADDRESS OF 1FD
                    ;14044
U 1016, FF12,15     ;14045      INC LS[T9]                          ;ADDRESS NOW HAS 10(B) IN LOW 2 BITS
U 1017, FF88,15     ;14046      INC LS[ADDRESS.DATA]                ;ADDRESS TO PRINT UNDER OTHER
U 1018, 0905,EC     ;14047      JSR [LOOP.T1C.1.2]                  ;DO TEST WITH ADDRESS OF 1FE
                    ;14048
U 1019, FF12,15     ;14049      INC LS[T9]                          ;ADDRESS NOW HAS 11(B) IN LOW 2 BITS
U 101A, FF88,15     ;14050      INC LS[ADDRESS.DATA]                ;ADDRESS TO PRINT UNDER OTHER
U 101B, 0905,EC     ;14051      JSR [LOOP.T1C.1.2]                  ;DO TEST WITH ADDRESS OF 1FF
                    ;14052
U 101C, FF82,15     ;14053      INC LS[ERROR.NUMBER]                ;ERROR 3
```

```
U 101D, FC12,15 ;14054          DEC LS[T9]                      ;ADDRESS NOW HAS 10(B) IN LOW 2 BITS
U 101E, FC88,15 ;14055          DEC LS[ADDRESS.DATA]            ;ADDRESS TO PRINT UNDER OTHER
                ;14056  LOOP.T1C.3:
U 101F, 2F82,95 ;14057          CLR WR[1]                       ;EXPECTED DATA (WR ACROSS PAGE ERR CLEAR)
U 1020, 9912,B5 ;14058          MEM.REQ[TEST.V.WCHK] ADRS[T9] DT[WORD]
                ;14059                                          ;SET UP TO TEST AT ALIGNED WORD
                ;14060                                          ; BOUNDARY
U 1021, 3022,15 ;14061          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE, BUT
                ;14062                                          ; DO NOT CHECK RESULT
U 1022, 9D45,75 ;14063          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;14064                                          ;SET UP TO READ CSR 1
U 1023, 3022,15 ;14065          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 1024, 0869,3C ;14066          JSR [CHECK.RESULT]              ;CHECK CSR 1 FOR WRITE ACROSS PAGE CLEAR
U 1025, 8901,F4 ;14067          JMP [LOOP.T1C.3]                ;ERROR LOOP IF ENABLED
                ;14068
U 1026, FF82,15 ;14069          INC LS[ERROR.NUMBER]            ;ERROR 4
U 1027, FF12,15 ;14070          INC LS[T9]                      ;ADDRESS NOW HAS 11(B) IN LOW 2 BITS
U 1028, FF88,15 ;14071          INC LS[ADDRESS.DATA]            ;ADDRESS TO PRINT UNDER OTHER
                ;14072  LOOP.T1C.4:
U 1029, 2F82,95 ;14073          CLR WR[1]                       ;EXPECTED DATA (WR ACROSS PAGE ERR CLEAR)
U 102A, 1912,95 ;14074          MEM.REQ[TEST.V.WCHK] ADRS[T9] DT[BYTE]
                ;14075                                          ;SET UP TO TEST AT BYTE BOUNDARY
U 102B, 3022,15 ;14076          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE, BUT
                ;14077                                          ; DO NOT CHECK RESULT
U 102C, 9D45,75 ;14078          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;14079                                          ;SET UP TO READ CSR 1
U 102D, 3022,15 ;14080          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 102E, 0869,3C ;14081          JSR [CHECK.RESULT]              ;CHECK CSR 1 FOR WRITE ACROSS PAGE CLEAR
U 102F, 8902,94 ;14082          JMP [LOOP.T1C.4]                ;ERROR LOOP IF ENABLED
                ;14083
U 1030, FF82,15 ;14084          INC LS[ERROR.NUMBER]            ;ERROR 5
U 1031, B613,15 ;14085          MOV LS[T9] TO WR[2]             ;BITS 0-8 SET
U 1032, 4543,15 ;14086          BIC LS[BIT1] TO WR[2]           ;CLEAR BIT 1
                ;14087  REPEAT.T1C.5:
U 1033, 23C1,15 ;14088          ROL WR[2]                       ;SHIFT PATTERN LEFT ONE BIT (SHIFTING
                ;14089                                          ; 0'S ACROSS BITS 2-8)
U 1034, C553,15 ;14090          BIC LS[BIT9] TO WR[2]           ;CLEAR BIT 9 IF SET
U 1035, 4741,15 ;14091          BIS LS[BIT0] TO WR[2]           ;SET BIT 0
U 1036, 3E13,15 ;14092          MOV WR[2] TO LS[T9]             ;PUT ADDRESS IN LS FOR MEM.REQ
U 1037, 3E89,15 ;14093          MOV WR[2] TO LS[ADDRESS.DATA]   ;ADDRESS TO PRINT UNDER OTHER
                ;14094  LOOP.T1C.5:
U 1038, 2F82,95 ;14095          CLR WR[1]                       ;EXPECTED DATA (WR ACROSS PAGE ERR CLEAR)
U 1039, 1912,F5 ;14096          MEM.REQ[TEST.V.WCHK] ADRS[T9] DT[LONG]
                ;14097                                          ;SET UP TO TEST AT UNALIGNED LONGWORD
                ;14098                                          ; BOUNDARY
U 103A, 3022,15 ;14099          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE, BUT
                ;14100                                          ; DO NOT CHECK RESULT
U 103B, 9D45,75 ;14101          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;14102                                          ;SET UP TO READ CSR 1
U 103C, 3022,15 ;14103          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 103D, 0869,3C ;14104          JSR [CHECK.RESULT]              ;CHECK CSR 1 FOR WRITE ACROSS PAGE CLEAR
U 103E, 8903,84 ;14105          JMP [LOOP.T1C.5]                ;ERROR LOOP IF ENABLED
                ;14106
                ;14107          BIT LS[BIT8] WITH WR[2],        ;SEE IF BIT 8 HAS BEEN CLEARED
U 103F, 5951,35 ;14108           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
```

```
U 1040, 0903.31 ;14109          JMP.IF[BIT.SET] TO [REPEAT.T1C.5] ;REPEAT WITH NEW PATTERN IF BIT 8 SET
                ;14110
U 1041, FF82.15 ;14111          INC LS[ERROR.NUMBER]              ;ERROR 6
U 1042, B626.15 ;14112          MOV LS[#FF] TO WR[0]              ;SET BITS 0-7
U 1043, 4750.15 ;14113          BIS LS[BIT8] TO WR[0]             ;NOW BITS 0-8 SET
U 1044, BE12.15 ;14114          MOV WR[0] TO LS[T9]               ;STORE IN LS FOR WRITE
U 1045, BE88.15 ;14115          MOV WR[0] TO LS[ADDRESS.DATA]     ;ADDRESS TO PRINT UNDER OTHER
U 1046, B676.15 ;14116          MOV LS[MME] TO WR[0]              ;SET MME BIT
U 1047, B67A.15 ;14117          MOV LS[TB.PAR.DIAG] TO WR[0]      ;AND TB PARITY DIAG BIT
U 1048, 8A17.FC ;14118          JSR [WRITE.CSR1]                  ;SET THESE BITS IN CSR 1
                ;14119  LOOP.T1C.6:
U 1049, 2F82.95 ;14120          CLR WR[1]                         ;EXPECTED DATA (WR ACROSS PAGE ERR CLEAR)
U 104A, 1B12.75 ;14121          MEM.REQ[WRITE.V.NOCHK] ADRS[T9] DT[LONG]
                ;14122                                            ;SET UP TO WRITE AT UNALIGNED LONGWORD
                ;14123                                            ; BOUNDARY WITH NOCHK
U 104B, B29C.15 ;14124          WRITE.MEM LS[ZERO]                ;WRITE ZEROS TO COMPLETE CYCLE
U 104C, 9D45.75 ;14125          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;14126                                            ;SET UP TO READ CSR 1
U 104D, 3022.15 ;14127          MOV MEM.DATA TO WR[0]             ;GET CONTENTS OF CSR 1
U 104E, 0869.3C ;14128          JSR [CHECK.RESULT]                ;CHECK CSR 1 FOR WRITE ACROSS PAGE CLEAR
U 104F, 8904.94 ;14129          JMP [LOOP.T1C.6]                  ;ERROR LOOP IF ENABLED
                ;14130
U 1050, FF82.15 ;14131          INC LS[ERROR.NUMBER]              ;ERROR 7
U 1051, B69E.15 ;14132          MOV LS[ONES] TO WR[0]             ;GET ALL ONES IN WR
U 1052, 457C.15 ;14133          BIC LS[BIT30] TO WR[0]            ;CLEAR BIT 30
U 1053, C57E.15 ;14134          BIC LS[BIT31] TO WR[0]            ;AND BIT 31.  NOW 0-29 ARE SET
U 1054, BE12.15 ;14135          MOV WR[0] TO LS[T9]               ;PUT ADDRESS IN LS FOR WRITE
U 1055, BE88.15 ;14136          MOV WR[0] TO LS[ADDRESS.DATA]     ;ADDRESS TO PRINT UNDER OTHER
                ;14137  LOOP.T1C.7:
U 1056, B666.95 ;14138          MOV LS[WR.ACROSS.PG] TO WR[1]     ;EXPECTED DATA (WRITE ACROSS PG ERR SET)
U 1057, 1B12.75 ;14139          MEM.REQ[WRITE.V.NOCHK] ADRS[T9] DT[LONG]
                ;14140                                            ;SET UP TO WRITE AT UNALIGNED LONGWORD
                ;14141                                            ; BOUNDARY WITH NOCHK AND SYS ADDR VIOL
U 1058, B29C.15 ;14142          WRITE.MEM LS[ZERO]                ;WRITE ZEROS TO COMPLETE CYCLE
U 1059, 9D45.75 ;14143          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;14144                                            ;SET UP TO READ CSR 1
U 105A, 3022.15 ;14145          MOV MEM.DATA TO WR[0]             ;GET CONTENTS OF CSR 1
U 105B, 0869.3C ;14146          JSR [CHECK.RESULT]                ;CHECK CSR 1 FOR WRITE ACROSS PAGE SET
U 105C, 0905.64 ;14147          JMP [LOOP.T1C.7]                  ;ERROR LOOP IF ENABLED
U 105D, 0906.64 ;14148          JMP [END.T1C]                     ;DONE
                ;14149
                ;14150
                ;14151  LOOP.T1C.1.2:
U 105E, 2006.95 ;14152          MOV WR[3] TO WR[1]                ;EXPECTED STATE OF WR ACROSS PG ERR BIT
U 105F, 1912.F5 ;14153          MEM.REQ[TEST.V.WCHK] ADRS[T9] DT[LONG]
                ;14154                                            ;SET UP TO TEST AT ALIGNED LONGWORD
                ;14155                                            ; BOUNDARY
U 1060, 3022.15 ;14156          MOV MEM.DATA TO WR[0]             ;ISSUE MOV TO COMPLETE U-CYCLE, BUT
                ;14157                                            ; DO NOT CHECK RESULT
U 1061, 9D45.75 ;14158          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;14159                                            ;SET UP TO READ CSR 1
U 1062, 3022.15 ;14160          MOV MEM.DATA TO WR[0]             ;GET CONTENTS OF CSR 1
U 1063, 0869.3C ;14161          JSR [CHECK.RESULT]                ;CHECK CSR 1 WRITE ACROSS PAGE ERR BIT
U 1064, 8905.E4 ;14162          JMP [LOOP.T1C.1.2]                ;ERROR LOOP IF ENABLED
U 1065, 5B00.14 ;14163          RETURN                            ;DONE  CHECK NEXT CONDITION
```

```
;14164
;14165  END.T1C:
```

```
;14166  .PAGE   " Main Memory Array Tests"
;14167
;14168  .toc    "  TEST 1D Size and Initialize Main Memory (M8391  MCT or M8728 ARRAY Module)"
;14169  ;++
;14170  ;
;14171  ; Test Description:
;14172  ;
;14173  ;       This test writes all of main memory with 0's on longword boundaries
;14174  ;       to initialize it.  Initialization is necessary to assure that the
;14175  ;       checkbits have been written correctly so that error correction will
;14176  ;       work on reads from main memory.  As the memory is being written, the
;14177  ;       ERROR SUM bit is checked.  When an error sum occurs, CSR 1 is read to
;14178  ;       assure that an NXM caused the error.  This is the top of main memory
;14179  ;       and this size is saved in an LS location for use in later tests.  The
;14180  ;       monitor will also print the memory size out in 256 KB blocks for veri-
;14181  ;       fication by the operator.  Under APT, the test will compare the computed
;14182  ;       size with that supplied by APT to assure that the logic used for sizing
;14183  ;       is functioning correctly.
;14184  ;         This test will also check for holes in memory by continuing to look
;14185  ;       for NXM on 256KB blocks after the first NXM is detected until address
;14186  ;       EC0000(H) is reached (the next 256KB block would be F00000(H) which is
;14187  ;       the UB ADAPTER REG space).
;14188  ;         Note:  A hole in memory will not cause the tests following this one
;14189  ;       to fail, however only contiguous memory will be tested.  Any memory
;14190  ;       above the hole will be ignored.
;14191  ;
;14192  ; Assumptions:
;14193  ;
;14194  ;       It is assumed that all previous tests have run successfully, especially
;14195  ;       the NXM logic test.  If NXM is not working, this test could hang.  If
;14196  ;       the NXM test passes, this test should not hang.
;14197  ;
;14198  ; Test Steps:
;14199  ;
;14200  ;       1)  Set up error number and module number in LS (for error printouts)
;14201  ;           and clear previous error number in LS.
;14202  ;       2)  Set up error mask to check bit 16 only (NXM bit in CSR 1).
;14203  ;       3)  Clear the LS location used to store the memory size (MM.LASTADDR+1).
;14204  ;       4)  Execute MEM.REQ with MF=WRITE.P and DT=longword.  Use the LS loc-
;14205  ;           ation MM.LASTADDR+1 as the address.
;14206  ;       5)  Execute WRITE.MEM LS using an LS location containing all zeros as
;14207  ;           the data.
;14208  ;       6)  Check for ERROR SUM and go to step 8 if an error is detected.
;14209  ;       7)  Increment MM.LASTADDR+1 by 4 and go to step 4.
;14210  ;       8)  Read CSR 1 and check that NXM is set.  If not, load the address
;14211  ;           being accessed into OTHER DATA, report an error, clear MM.LAST.ADDR
;14212  ;           +1, and abort this test.  Else continue.
;14213  ;       9)  Check that MM.LAST.ADDR+1 is on a 256KB boundary.  Report error and
;14214  ;           abort test if not.  Else continue.
;14215  ;       10) Convert value in MM.LASTADDR+1 to number of 256KB blocks and put
;14216  ;           in LS 7.  Issue CPU.ATTN to console to print this value.
;14217  ;       11) If under APT control, compare APT supplied memory size with that
;14218  ;           found in this test.  Report error if not the same.
;14219  ;       12) Change module code to indicate ARRAY board if an error occurs.
;14220  ;       13) Store MM.LASTADDR+1 in a temporary LS location.  Repeat steps 4 and
```

```
;14221  ;           5 except us the temporary LS location as the address.
;14222  ;       14) Read CSR 1 and check that NXM is set.  Load the address being ac-
;14223  ;           cessed into OTHER DATA and report an error if not (there is a hole
;14224  ;           below this address).
;14225  ;       15) Increment the temporary LS location by 40000(H) (256KB) and repeat
;14226  ;           steps 11 through 13 until the address EC0000(H) has been accessed.
;14227  ;
;14228  ; Errors:
;14229  ;
;14230  ;       Note:  Data under OTHER in error report is address in main memory that
;14231  ;              failed.
;14232  ;
;14233  ;       Error 1 - ERROR SUM occurred before NXM was reached. EXP data is CSR 1
;14234  ;                 NXM set, RECV data is actual NXM bit in CSR 1.
;14235  ;       Error 2 - NXM occured, but not on a 256KB boundary.  EXP data is all
;14236  ;                 0's, RECV data is address with bits 18-31 cleared.
;14237  ;       Error 3 - Memory size does not compare with what APT supplied (this
;14238  ;                 error can only occur when under APT control).  EXP data is
;14239  ;                 APT supplied size in 256K byte blocks, RECV is size found
;14240  ;                 by test.
;14241  ;       Error 4 - Memory is holey.  NXM failed to set at address above where it
;14242  ;                 set before.  EXP data is CSR 1 NXM set, RECV data is actual
;14243  ;                 NXM bit in CSR 1.
;14244  ;
;14245  ; Debug:
;14246  ;
;14247  ;       NOTE:  If the MCT is single stepped, refresh on the ARRAY module will
;14248  ;       be lost.  Thus the array data will be lost.
;14249  ;
;14250  ;       Two methods of debug will be explained.  If the MCT module is in a
;14251  ;       test station then the MCT can be single stepped making debug easier
;14252  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;14253  ;       method will be explained also.  When single stepping the memory con-
;14254  ;       troller will help in debug, this section will explicitly suggest it.
;14255  ;
;14256  ;       Error 1 - This error indicates a problem with the CSR error logic.  Run
;14257  ;       the previous tests again to try to pinpoint the problem.  The CSR can
;14258  ;       be examined with the console EX MC 1 command to see which error bit was
;14259  ;       asserted.  The address being accessed when the error occurred will be
;14260  ;       printed under OTHER DATA in the error report.
;14261  ;
;14262  ;       Error 2 - This error indicates that NXM occurred before reaching a 256
;14263  ;       byte boundary.  Noise in the NXM logic is a possibility.  Check for
;14264  ;       noise while looping on error.  OTHER data in the error printout con-
;14265  ;       tains the address where the NXM occurred.
;14266  ;
;14267  ;       Error 3 - This error can only occur under APT control.  It indicates
;14268  ;       that the size supplied by APT in the mailbox area did not compare with
;14269  ;       the size computed by this test.  The data under EXPECTED is the number
;14270  ;       of 256K byte blocks that APT said should be found.  The number under
;14271  ;       RECEIVED is the number of 256K byte blocks found by this test.  Suspect
;14272  ;       the DECODER A or DECODER B PALS if this test fails.
;14273  ;
;14274  ;       Error 4 - This error indicates that a hole exists in main memory.  The
;14275  ;       address under 'OTHER' in the error report is the address at which an
```

```
                    ;14276  ;       NXM failed to get set even though NXM had set at a lower address.  The
                    ;14277  ;       most likely problem is a jumper on the array board is wrong or one of
                    ;14278  ;       higher PA bits is stuck low.
                    ;14279  ;
                    ;14280  ;--
                    ;14281
                    ;14282  T.1D:
                    ;14283
U 1066, B65E,15     ;14284          MOV LS[BEGIN.TEST] TO WR[0]      ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;14285                                           ; STATUS WORD
U 1067, 3E80,15     ;14286          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;14287                                           ; BIT 15 INDICATES START OF TEST TO
                    ;14288                                           ; CONSOLE
U 1068, 10E0,15     ;14289          MISC [SET.CP.ATTN]               ;ISSUE CPU ATTN TO CONSOLE
                    ;14290  WAIT.T1D.0:
U 1069, 0906,94     ;14291          JMP [WAIT.T1D.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                    ;14292                                           ; RESPOND
                    ;14293
U 106A, 0A1A,9C     ;14294          JSR [SETUP]                      ;SET UP MASKS, MODULE CODE ETC. AND
                    ;14295                                           ; CLEAR MCT CSR 1
                    ;14296
U 106B, B670,15     ;14297          MOV LS[OTHER.DATA] TO WR[0]      ;SET UP WR TO WRITE CONTROL WORD
U 106C, 3E80,15     ;14298          MOV WR[0] TO LS[CONTROL.STATUS] ;CONSOLE WILL PRINT ADDRESS UNDER 'OTHER''
U 106D, 6524,15     ;14299          CLR LS[MM.LASTADDR+1]            ;CLEAR AN LS LOCATION (LOCATION 12(H))
                    ;14300                                           ; FOR STORAGE OF ADDRESS
U 106E, 6588,15     ;14301          CLR LS[ADDRESS.DATA]             ;ADDRESS TO PRINT IF ERROR
U 106F, B645,15     ;14302          MOV LS[#4] TO WR[2]              ;INCREMENT VALUE FOR MEM ADDRESS
                    ;14303  LOOP.T1D.1:
U 1070, 5F60,15     ;14304          MCOM LS[NXM] TO WR[0]            ;CLEAR BIT 16 IN WR
U 1071, 3E8A,15     ;14305          MOV WR[0] TO LS[ERROR.MASK]      ;CHECK NXM BIT IN CSR 1 ONLY
U 1072, B640,15     ;14306          MOV LS[#1] TO WR[0]              ;1 IN WR
U 1073, BE82,15     ;14307          MOV WR[0] TO LS[ERROR.NUMBER]    ;SET UP ERROR NUMBER IN LS
                    ;14308  REPEAT.T1D.1:
U 1074, 9924,75     ;14309          MEM.REQ[WRITE.P] ADRS[MM.LASTADDR+1] DT[LONG]
                    ;14310                                           ;SET UP TO WRITE 0'S THROUGHOUT MEMORY
U 1075, B29C,15     ;14311          WRITE.MEM LS[ZERO]               ;WRITE ALL ZEROS IN MEMORY
U 1076, 5B00,1D     ;14312          SKIP.IF[MEM.REF.OK]              ;SKIP NEXT INTRUCTION IF NO ERROR SUM
U 1077, 8907,A4     ;14313          JMP [T1D.CHECK.NXM]              ;JMP TO CHECK THAT ERROR SUM WAS AN NXM
U 1078, EC25,15     ;14314          ADD WR[2] TO LS[MM.LASTADDR+1]   ;INCREMENT MEMORY ADDRESS BY 4
U 1079, 0907,44     ;14315          JMP [REPEAT.T1D.1]               ;REPEAT UNTIL GET ERROR SUM
                    ;14316
                    ;14317  T1D.CHECK.NXM:
U 107A, 3624,15     ;14318          MOV LS[MM.LASTADDR+1] TO WR[0]   ;GET ADDRESS
U 107B, BE88,15     ;14319          MOV WR[0] TO LS[ADDRESS.DATA]    ;SET UP TO PRINT IF ERROR
U 107C, B660,95     ;14320          MOV LS[NXM] TO WR[1]             ;EXPECTED DATA (NXM SET)
U 107D, 9D45,75     ;14321          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;14322                                           ;SET UP TO READ CSR 1
U 107E, 3022,15     ;14323          MOV MEM.DATA TO WR[0]            ;GET CONTENTS OF CSR 1
U 107F, 0869,3C     ;14324          JSR [CHECK.RESULT]               ;CHECK FOR NXM SET
U 1080, 0907,44     ;14325          JMP [REPEAT.T1D.1]               ;ERROR LOOP IF ENABLED
                    ;14326
U 1081, FF82,15     ;14327          INC LS[ERROR.NUMBER]             ;ERROR 2
U 1082, E58A,15     ;14328          CLR LS[ERROR.MASK]               ;CHECK ALL BITS
U 1083, 3624,15     ;14329          MOV LS[MM.LASTADDR+1] TO WR[0]   ;GET VALUE OF LAST ADDRESS + 1
U 1084, DF28,95     ;14330          MCOM LS[#FFFF] TO WR[1]          ;SET BITS 16-31 IN WR
```

```
U 1085, 4560,95 ;14331          BIC LS[BIT16] TO WR[1]          ;CLEAR BIT 16
U 1086, C562,95 ;14332          BIC LS[BIT17] TO WR[1]          ;CLEAR BIT 17. NOW BITS 18-31 SET
U 1087, AF42,15 ;14333          BIC WR[1] TO WR[0]              ;CLEAR BITS 18-31 OF LAST ADDRESS+1
U 1088, 2F82,95 ;14334          CLR WR[1]                       ;EXPECTED DATA (ALL ZEROS INDICATES LAST
                ;14335                                          ; ADDRESS+1 IS ON 256KB BOUNDARY)
U 1089, 0869,3C ;14336          JSR [CHECK.RESULT]              ;CHECK THAT ADDRESS IS ON 256KB BOUNDARY
U 108A, 8907,04 ;14337          JMP [LOOP.T1D.1]                ;ERROR LOOP IF ENABLED
                ;14338
U 108B, B680,15 ;14339          MOV LS[CONTROL.STATUS] TO WR[0] ;GET CONTROL AND STATUS WORD
                ;14340          BIT LS[ERROR] WITH WR[0],       ;SEE IF AN ERROR OCCURRED ON SIZING
U 108C, D950,35 ;14341           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 108D, 8909,09 ;14342          JMP.IF[BITS.CLR] TO [T1D.PRINT.SIZE] ;GO PRINT SIZE OF MEMORY IF NO
                ;14343                                          ; SIZEING ERROR
U 108E, 6524,15 ;14344          CLR LS[MM.LASTADDR+1]           ;ELSE CLEAR MM.LASTADDR+1 SO FUTURE
                ;14345                                          ; MEMORY TESTS WILL HAVE TO RESIZE
U 108F, 890B,A4 ;14346          JMP [END.T1D]                   ;AND ABORT REST OF THIS TEST
                ;14347
                ;14348  T1D.PRINT.SIZE:
U 1090, B625,15 ;14349          MOV LS[MM.LASTADDR+1] TO WR[2]  ;PUT LASTADDR+1 IN WR
U 1091, 3E13,15 ;14350          MOV WR[2] TO LS[T9]             ;SAVE LAST ADDRESS + 1 IN TEMP LS FOR
                ;14351                                          ; HOLE TEST
U 1092, 3648,15 ;14352          MOV LS[#10] TO WR[0]            ;16 DECIMAL IN WR 0
U 1093, C042,15 ;14353          ADD LS[#2] TO WR[0]             ;18 DECIMAL IN WR 0
                ;14354  T1D.SHIFT.LOOP:
U 1094, A341,15 ;14355          ROR WR[2]                       ;SHIFT LAST ADDRESS+1 RIGHT
                ;14356          DEC WR[0],                      ;DECREMENT COUNTER
U 1095, A100,35 ;14357           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1096, 8909,41 ;14358          JMP.IF[NEQ] TO [T1D.SHIFT.LOOP] ;REPEAT UNTIL SHIFTED RIGHT 18 PLACES
                ;14359
U 1097, BE0F,15 ;14360          MOV WR[2] TO LS[MEMORY.SIZE]    ;PUT SIZE IN 256KB BLOCKS IN LS 7 FOR
                ;14361                                          ; PRINTING
U 1098, 3656,15 ;14362          MOV LS[PRINT.MEM.SIZE] TO WR[0] ;SET BIT 11 IN WR
U 1099, 3E80,15 ;14363          MOV WR[0] TO LS[CONTROL.STATUS] ;INFORMS CONSOLE TO PRINT MEMORY SIZE
U 109A, 10E0,15 ;14364          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;14365  WAIT.T1D.1:
U 109B, 8909,B4 ;14366          JMP [WAIT.T1D.1]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;14367                                          ; RESPOND
                ;14368
U 109C, FF82,15 ;14369          INC LS[ERROR.NUMBER]            ;ERROR 3 (APT ONLY)
U 109D, 3690,15 ;14370          MOV LS[ERROR.CONTROL] TO WR[0]  ;GET ERROR CONTROL WORD
                ;14371          BIT LS[APT.PRESENT] WITH WR[0], ;SEE IF UNDER APT
U 109E, 594A,35 ;14372           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 109F, 890A,59 ;14373          JMP.IF[BITS.CLR] TO [T1D.HOLE.CHECK] ;IF NOT APT, GO CHECK FOR HOLEY
                ;14374                                          ; MEMORY
U 10A0, 3692,95 ;14375          MOV LS[APT.PARAM] TO WR[1]      ;GET APT PARAMATERS
U 10A1, C52C,95 ;14376          BIC LS[#FFFFFF00] TO WR[1]      ;CLEAR ALL BUT LOW BYTE (SIZE PARAM)
                ;14377                                          ; (EXPECTED DATA)
U 10A2, 2004,15 ;14378          MOV WR[2] TO WR[0]              ;SIZE FOUND BY TEST IN 256K BLOCKS
U 10A3, 0869,3C ;14379          JSR [CHECK.RESULT]              ;SEE IF SIZE COMPARES TO THAT SUPPLIED
                ;14380                                          ; BY APT
U 10A4, DB00,15 ;14381          NOP                             ;NO ERROR LOOP UNDER APT
                ;14382
                ;14383  T1D.HOLE.CHECK:
U 10A5, FF82,15 ;14384          INC LS[ERROR.NUMBER]            ;ERROR 4
U 10A6, 5F60,15 ;14385          MCOM LS[NXM] TO WR[0]           ;CLEAR BIT 16 IN WR
```

```
U 10A7, 3E8A,15 ;14386          MOV WR[0] TO LS[ERROR.MASK]      ;CHECK NXM BIT IN CSR 1 ONLY
U 10A8, B670,15 ;14387          MOV LS[OTHER.DATA] TO WR[0]      ;SET UP WR TO WRITE CONTROL WORD
U 10A9, 3E80,15 ;14388          MOV WR[0] TO LS[CONTROL.STATUS] ;CONSOLE WILL PRINT ADDRESS UNDER 'OTHER''
U 10AA, 3648,15 ;14389          MOV LS[ARRAY] TO WR[0]           ;GET MODULE CODE FOR ARRAY BOARD
U 10AB, 3E8C,15 ;14390          MOV WR[0] TO LS[MODULE.NUM]      ;PRINT ARRAY AS FAILING MODULE IF ERROR
                ;14391  LOOP.T1D.4:
U 10AC, B660,95 ;14392          MOV LS[NXM] TO WR[1]             ;EXPECTED DATA (NXM SET)
U 10AD, 9912,75 ;14393          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;14394                                           ;SET UP TO WRITE TO NON-EXISTANT MEMORY
U 10AE, B29C,15 ;14395          WRITE.MEM LS[ZERO]               ;WRITE ZEROS
U 10AF, 9D45,75 ;14396          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;14397                                           ;SET UP TO READ CSR 1
U 10B0, 3022,15 ;14398          MOV MEM.DATA TO WR[0]            ;GET CONTENTS OF CSR 1
U 10B1, 0869,3C ;14399          JSR [CHECK.RESULT]               ;CHECK THAT NXM IS SET
U 10B2, 090A,C4 ;14400          JMP [LOOP.T1D.4]                 ;ERROR LOOP IF ENABLED
                ;14401
U 10B3, 3612,15 ;14402          MOV LS[T9] TO WR[0]              ;GET CURRENT ADDRESS
U 10B4, 4064,15 ;14403          ADD LS[#40000] TO WR[0]          ;ADD 256KB TO ACCESS NEXT BLOCK
U 10B5, BE12,15 ;14404          MOV WR[0] TO LS[T9]              ;STORE NEW ADDRESS IN LS
U 10B6, BE88,15 ;14405          MOV WR[0] TO LS[ADDRESS.DATA]    ;ALSO ADDRESS TO PRINT IF ERROR
U 10B7, 373A,95 ;14406          MOV LS[#F00000] TO WR[1]         ;SEE IF IN UNIBUS ADAPTER SPACE
                ;14407          CMP WR[1] WITH WR[0],            ;SEE IF ADDRESS = F00000(H)
U 10B8, 2642,35 ;14408           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 10B9, 090A,C1 ;14409          JMP.IF[NEQ] TO [LOOP.T1D.4]      ;REPEAT IF NOT UNTIL DONE
                ;14410  END.T1D:
```

```
;14411  .PAGE   "  TEST 1E All 1's and All 0's Data Test (M8728  ARRAY Module, M8391  MCT Module)"
;14412  ;++
;14413  ;
;14414  ; Test Description:
;14415  ;
;14416  ;       This test checks all available memory for the ability to write and read
;14417  ;       all ones and write and read all zeros.  A data pattern of 80000000(H)
;14418  ;       is also run as this pattern complements the CKBTs in memory.  Any un-
;14419  ;       correctable errors (2 or more bits in a single longword) will produce
;14420  ;       a normal error message with the address of the failure under OTHER DATA
;14421  ;       in the error report.  The test will write all 1's and read all ones
;14422  ;       using WRITE.P and READ.P with DT=longword.  After checking for errors
;14423  ;       it will repeat the sequence using data of 0's and data of 80000000(H).
;14424  ;       This is repeated for each address in succession. Since the test is run
;14425  ;       with error correction enabled, only uncorrectable errors will be detec-
;14426  ;       ted on the data comparison.  These errors will generate an error re-
;14427  ;       port.
;14428  ;         Note that this is not an address test.  A failure in an address line
;14429  ;       will not cause this test to fail.
;14430  ;
;14431  ; Assumptions:
;14432  ;
;14433  ;       It is assumed that all previous tests have run successfully.
;14434  ;
;14435  ; Test Steps:
;14436  ;
;14437  ;       1)  Set up error mask, error number, and module number in LS (for
;14438  ;           error printouts) and clear previous error number in LS.
;14439  ;       2)  Clear CSR 1 control bits, load a temporary LS location with 0's
;14440  ;           (will be used as the address) and load LS MM.LASTADDR into WR 2.
;14441  ;       3)  Check that WR 2 (last memory address+1) is non-zero.  Size memory
;14442  ;           and report size if not sized previously.
;14443  ;       4)  Execute MEM.REQ with MF=WRITE.P and DT=longword.  Use the tempor-
;14444  ;           ary LS location loaded previously as the address.
;14445  ;       5)  Execute WRITE.MEM LS using 1's as the data.
;14446  ;       6)  Execute MEM.REQ with MF=READ.P and MF=longword.  Use the temporary
;14447  ;           LS location as the address.
;14448  ;       7)  Execute MOV MEM.DATA TO WR[0] and check for ones.  Report an error
;14449  ;           if this fails. Go to step 10.
;14450  ;       8)  Repeat steps 4 through 7 with data of 0's.  In step 7 if no error
;14451  ;           is detected in the data, check for an ERROR SUM.  If error sum,
;14452  ;           read CSR 1 and check the RDS bit.  Report an error if set (CKBT
;14453  ;           failure).
;14454  ;       9)  Repeat step 8 with data of 80000000(H).
;14455  ;       10) Increment the temporary LS location containing the address by 4.
;14456  ;       11) Compare the new address with the last memory address + 1.  If not
;14457  ;           equal, go to step 4.  Otherwise test is done.
;14458  ;
;14459  ; Errors:
;14460  ;
;14461  ;       NOTE:  Other data in following errors is memory address that failed.
;14462  ;
;14463  ;       Error 1 - Uncorrectable memory data error with data of all ones.
;14464  ;       Error 2 - Uncorrectable memory data error with data of all zeros.
;14465  ;       Error 3 - Uncorrectable error in CKBTS indicated by RDS being set in
```

```
                ;14466  ;                   CSR 1.  CKBT data of 0111100(B).
                ;14467  ;       Error 4 - Uncorrectable memory data error with data of 80000000(H).
                ;14468  ;       Error 5 - Uncorrectable error in CKBTS indicated by RDS being set in
                ;14469  ;                   CSR 1.  CKBT data of 1000011(B).
                ;14470  ;
                ;14471  ; Debug:
                ;14472  ;
                ;14473  ;       NOTE:  If the MCT is single stepped, refresh on the ARRAY module will
                ;14474  ;       be lost.  Thus the array data will be lost.
                ;14475  ;
                ;14476  ;       Two methods of debug will be explained.  If the MCT module is in a
                ;14477  ;       test station then the MCT can be single stepped making debug easier
                ;14478  ;       in some cases.  Otherwise only the CPU can be single stepped and this
                ;14479  ;       method will be explained also.  When single stepping the memory con-
                ;14480  ;       troller will help in debug, this section will explicitly suggest it.
                ;14481  ;
                ;14482  ;       Error 1 - This error indicates 2 or more bad data bits on the array
                ;14483  ;       or a problem with the write/read logic on the MCT module.  If address
                ;14484  ;       0 is failing, check the signal MEM SEL A L from the DECODER A PAL on
                ;14485  ;       the MCT module.  This can be checked after single stepping the CPU to
                ;14486  ;       the MEM.REQ with MF=WRITE.P.  The MEM SEL A L signal should be low.  If
                ;14487  ;       not, the DECODER A PAL is probably bad.  This would also apply to ad-
                ;14488  ;       dresses which cross a module boundary for the first time (e.g. address
                ;14489  ;       40000 on a 16K ram module or address 100000 on a 64K ram module).
                ;14490  ;         Other signals that should be checked if address 0 fails are CAS TIM
                ;14491  ;       L, RAS TIM L, WRT TIM L (during the write cycle), or DR EN L (during
                ;14492  ;       the read cycle).  These should be checked while looping on error.  The
                ;14493  ;       data lines themselves should also be checked to make sure they are get-
                ;14494  ;       ting off the board.  If all these signals seem to be working, the prob-
                ;14495  ;       lem is probably on the array module.
                ;14496  ;         If the error occurs on an address other than the first address on the
                ;14497  ;       module, the most likely suspect is a RAM chip.  The chip can be located
                ;14498  ;       from the address of the error (under OTHER DATA in the error report)
                ;14499  ;       and the position of the failing bit from the expected and received data
                ;14500  ;       in the error report.
                ;14501  ;
                ;14502  ;       Error 2 - Same as error 1.
                ;14503  ;
                ;14504  ;       Error 3 - This error indicates an uncorrectable CKBT failure.  The data
                ;14505  ;       coming back is correct, but the RDS bit is set, indicating that a dou-
                ;14506  ;       ble bit error was detected.  The most likely suspect is a RAM chip in
                ;14507  ;       the CKBT portion of the 39 bit word.  The CKBTs can be checked while
                ;14508  ;       looping on error. They should be 0111100(B) from CT to C1 respectively.
                ;14509  ;
                ;14510  ;       Error 4 - Same as error 1.  This error should not occur at addresses
                ;14511  ;       other than those detected in either error 1 or error 2.
                ;14512  ;
                ;14513  ;       Error 5 - Same as error 3 except the correct CKBTs are 1000011(B).
                ;14514  ;
                ;14515  ;--
                ;14516
                ;14517  T.1E:
                ;14518
U 10BA, B65E.15 ;14519          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;14520                                          ; STATUS WORD
```

```
U 10BB, 3E80,15 ;14521          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;14522                                         ; BIT 15 INDICATES START OF TEST TO
                ;14523                                         ; CONSOLE
U 10BC, 10E0,15 ;14524          MISC [SET.CP.ATTN]             ;ISSUE CPU ATTN TO CONSOLE
                ;14525  WAIT.T1E.0:
U 10BD, 090B,D4 ;14526          JMP [WAIT.T1E.0]               ;LOOP HERE WAITING FOR CONSOLE TO
                ;14527                                         ; RESPOND
                ;14528
U 10BE, 0A1A,9C ;14529          JSR [SETUP]                    ;SET UP MASKS, MODULE CODE ETC. AND
                ;14530                                         ; CLEAR MCT CSR 1
U 10BF, 4748,15 ;14531          BIS LS[ARRAY] TO WR[0]         ;ADD ARRAY BOARD TO MODULE CODE
U 10C0, 3E8C,15 ;14532          MOV WR[0] TO LS[MODULE.NUM]    ;STORE MODULE CODE IN LS TO INDICATE
                ;14533                                         ; MCT OR ARRAY BOARD
                ;14534
                ;14535          MOV LS[MM.LASTADDR+1] TO WR[2], ;GET LAST ADDRESS + 1 FROM LS
U 10C1, 3625,35 ;14536           DT(LONG)&SET.ALU.CC           ; AND SET CONDITION CODES
U 10C2, 890C,41 ;14537          JMP.IF[NEQ] TO [T1E.CONTINUE]  ;CONTINUE TESTING IF LAST ADDRESS+1 IS
                ;14538                                         ; NON-ZERO
U 10C3, 8A19,4C ;14539          JSR [SIZE.MEMORY]              ;ELSE SIZE MEMORY, PRINT RESULT, AND
                ;14540                                         ; CONTINUE
                ;14541
                ;14542  T1E.CONTINUE:
U 10C4, 3645,95 ;14543          MOV LS[#4] TO WR[3]            ;STORE INCREMENT VALUE IN WR 3
U 10C5, B670,15 ;14544          MOV LS[OTHER.DATA] TO WR[0]    ;SET UP TO PRINT UNDER 'OTHER' DATA
U 10C6, 3E80,15 ;14545          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL WORD
U 10C7, 6512,15 ;14546          CLR LS[T9]                     ;USED AS MAIN MEMORY PHYSICAL ADDRESS
U 10C8, 6588,15 ;14547          CLR LS[ADDRESS.DATA]           ;CLEAR ADDRESS TO PRINT IF ERROR
                ;14548  LOOP.T1E.1:
U 10C9, 369E,95 ;14549          MOV LS[ONES] TO WR[1]          ;EXPECTED DATA
U 10CA, 9912,75 ;14550          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;14551                                         ;SET UP TO WRITE DATA IN MEMORY
U 10CB, 0A1B,4C ;14552          JSR [NOP.DELAY]                ;USED TO PREVENT TIMEOUT IF ARRAY IS
                ;14553                                         ; ERATIC (DELAYS 5 CPU STATES)
U 10CC, 329E,15 ;14554          WRITE.MEM LS[ONES]             ;WRITE ONES IN CURRENT MEMORY LOCATION
U 10CD, 1813,F5 ;14555          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;14556                                         ;SET UP TO READ DATA BACK
U 10CE, 3022,15 ;14557          MOV MEM.DATA TO WR[0]          ;GET RESULT
U 10CF, 0869,3C ;14558          JSR [CHECK.RESULT]             ;CHECK DATA FOR ONES
U 10D0, 090C,94 ;14559          JMP [LOOP.T1E.1]               ;ERROR LOOP IF ENABLED
                ;14560
U 10D1, 6C13,95 ;14561          ADD WR[3] TO LS[T9]            ;INCREMENT CURRENT ADDRESS BY 4
U 10D2, 6C89,95 ;14562          ADD WR[3] TO LS[ADDRESS.DATA]  ;INCREMENT ADDRESS TO PRINT
                ;14563          CMP WR[2] WITH LS[T9],         ;SEE IF NEW ADDRESS = LAST ADDRESS + 1
U 10D3, CF13,35 ;14564           DT(LONG)&SET.ALU.CC           ; AND SET CONDITION CODES
U 10D4, 090C,91 ;14565          JMP.IF[NEQ] TO [LOOP.T1E.1]    ;REPEAT UNTIL ALL AVAILABLE MEMORY TESTED
                ;14566
U 10D5, E580,15 ;14567          CLR LS[CONTROL.STATUS]         ;CLEAR CONTROL AND STATUS WORD (NOP)
U 10D6, 10E0,15 ;14568          MISC [SET.CP.ATTN]             ;ISSUE CPU ATTN TO CONSOLE TO INFORM
                ;14569                                         ; HIM THAT CPU IS ALIVE
                ;14570  WAIT.T1E.1:
U 10D7, 090D,74 ;14571          JMP [WAIT.T1E.1]               ;LOOP HERE WAITING FOR CONSOLE TO
                ;14572                                         ; RESPOND
U 10D8, B670,15 ;14573          MOV LS[OTHER.DATA] TO WR[0]    ;SET UP TO PRINT UNDER 'OTHER' DATA
U 10D9, 3E80,15 ;14574          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL WORD
                ;14575
```

```
U 10DA, FF82,15 ;14576          INC LS[ERROR.NUMBER]              ;ERROR 2
U 10DB, 6512,15 ;14577          CLR LS[T9]                       ;MEMORY ADDRESS 0 IS FIRST
U 10DC, 6588,15 ;14578          CLR LS[ADDRESS.DATA]             ;SAME WITH ADDRESS TO PRINT IF ERROR
U 10DD, 5B00,1E ;14579          SKIP                             ;NEXT INSTRUCTION ONLY FOR LOOP ON
                ;14580                                           ; ERROR 3
                ;14581  LOOP.T1E.3:
U 10DE, FC82,15 ;14582          DEC LS[ERROR.NUMBER]             ;LOOP ON ERROR 4 ENTRY. DECREMENT ERROR
                ;14583                                           ; NUMBER TO ERROR 2
U 10DF, E58A,15 ;14584          CLR LS[ERROR.MASK]               ;CHECK ALL BITS
                ;14585  LOOP.T1E.2:
U 10E0, 2F82,95 ;14586          CLR WR[1]                        ;EXPECTED MEMORY DATA IS 0'S
U 10E1, 9912,75 ;14587          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;14588                                           ;SET UP TO WRITE MEMORY
U 10E2, B29C,15 ;14589          WRITE.MEM LS[ZERO]               ;WRITE CURRENT ADDRESS WITH ZEROS
U 10E3, 1813,F5 ;14590          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;14591                                           ;SET UP TO READ MEMORY
U 10E4, 0A1B,4C ;14592          JSR [NOP.DELAY]                  ;USED TO PREVENT TIMEOUT IF ARRAY IS
                ;14593                                           ; ERATIC (DELAYS 5 CPU STATES)
U 10E5, 3022,15 ;14594          MOV MEM.DATA TO WR[0]            ;GET DATA FROM MEMORY
U 10E6, 0869,3C ;14595          JSR [CHECK.RESULT]               ;CHECK RESULT FOR ZEROS
U 10E7, 890E,04 ;14596          JMP [LOOP.T1E.2]                 ;ERROR LOOP IF ENABLED
                ;14597
U 10E8, 5B00,1D ;14598          SKIP.IF[MEM.REF.OK]              ;SKIP NEXT INSTRUCTION IF NO ERROR SUM
U 10E9, 890E,F4 ;14599          JMP [TEST.T1E.3]                 ;ELSE GO CHECK CSR FOR RDS
                ;14600  TEST.T1E.2.INC:
U 10EA, 6C13,95 ;14601          ADD WR[3] TO LS[T9]               ;INCREMENT CURRENT ADDRESS BY 4
U 10EB, 6C89,95 ;14602          ADD WR[3] TO LS[ADDRESS.DATA]    ;ALSO INCREMENT ADDRESS TO PRINT
                ;14603          CMP WR[2] WITH LS[T9],           ;COMPARE NEW ADDRESS WITH LAST ADDRESS + 1
U 10EC, CF13,35 ;14604           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 10ED, 890E,01 ;14605          JMP.IF[NEQ] TO [LOOP.T1E.2]      ;REPEAT UNTIL ALL ADDRESS CHECKED
U 10EE, 090F,A4 ;14606          JMP [TEST.T1E.4]                 ;GO TO NEXT TEST IF DONE ALL ADDRESSES
                ;14607
                ;14608  TEST.T1E.3:
U 10EF, FF82,15 ;14609          INC LS[ERROP.NUMBER]             ;ERROR 3
U 10F0, 5F7E,15 ;14610          MCOM LS[RDS] TO WR[0]            ;CLEAR RDS (BIT 31) IN WR
U 10F1, 3E8A,15 ;14611          MOV WR[0] TO LS[ERROR.MASK]      ;CHECK RDS BIT ONLY IN CSR 1
U 10F2, 2F82,95 ;14612          CLR WR[1]                        ;EXPECTED DATA (RDS CLEAR)
U 10F3, 9D45,75 ;14613          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;14614                                           ;SET UP TO READ CSR 1
U 10F4, 3022,15 ;14615          MOV MEM.DATA TO WR[0]            ;GET CSR 1 CONTENTS
U 10F5, 0869,3C ;14616          JSR [CHECK.RESULT]               ;SEE IF RDS IS SET
U 10F6, 090D,E4 ;14617          JMP [LOOP.T1E.3]                 ;ERROR LOOP IF ENABLED
U 10F7, FC82,15 ;14618          DEC LS[ERROR.NUMBER]             ;RESTORE ERROR NUMBER TO ERROR 2
U 10F8, E58A,15 ;14619          CLR LS[ERROR.MASK]               ;RESTORE MASK TO CHECK ALL BITS
U 10F9, 890E,A4 ;14620          JMP [TEST.T1E.2.INC]             ;GO TO GENERATE NEXT ADDRESS
                ;14621
                ;14622  TEST.T1E.4:
U 10FA, E580,15 ;14623          CLR LS[CONTROL.STATUS]           ;CLEAR CONTROL AND STATUS WORD (NOP)
U 10FB, 10E0,15 ;14624          MISC [SET.CP.ATTN]               ;ISSUE CPU ATTN TO CONSOLE TO INFORM
                ;14625                                           ; HIM THAT CPU IS ALIVE
                ;14626  WAIT.T1E.2:
U 10FC, 090F,C4 ;14627          JMP [WAIT.T1E.2]                 ;LOOP HERE WAITING FOR CONSOLE TO
                ;14628                                           ; RESPOND
U 10FD, B670,15 ;14629          MOV LS[OTHER.DATA] TO WR[0]      ;SET UP TO PRINT UNDER 'OTHER' DATA
U 10FE, 3E80,15 ;14630          MOV WR[0] TO LS[CONTROL.STATUS]  ;SET UP CONTROL WORD
```

```
                ;14631
U 10FF, FF82,15 ;14632          INC LS[ERROR.NUMBER]            ;ERROR NUMBER IS NOW 3
U 1100, FF82,15 ;14633          INC LS[ERROR.NUMBER]            ;CURRENT ERROR NUMBER IS 4
U 1101, 6512,15 ;14634          CLR LS[T9]                      ;MEMORY ADDRESS 0 IS FIRST
U 1102, 6588,15 ;14635          CLR LS[ADDRESS.DATA]            ;SAME WITH ADDRESS TO PRINT IF ERROR
U 1103, 5B00,1E ;14636          SKIP                            ;NEXT INSTRUCTION ONLY FOR LOOP ON
                ;14637                                          ; ERROR 5
                ;14638  LOOP.T1E.5:
U 1104, FC82,15 ;14639          DEC LS[ERROR.NUMBER]            ;LOOP ON ERROR 5 ENTRY. DECREMENT ERROR
                ;14640                                          ; NUMBER TO ERROR 4
U 1105, E58A,15 ;14641          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
                ;14642  LOOP.T1E.4:
U 1106, B67E,95 ;14643          MOV LS[#80000000] TO WR[1]      ;EXPECTED DATA
U 1107, 9912,75 ;14644          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;14645                                          ;SET UP TO WRITE MEMORY
U 1108, B27E,15 ;14646          WRITE.MEM LS[#80000000]         ;WRITE CURRENT ADDRESS WITH 80000000(H)
U 1109, 1813,F5 ;14647          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;14648                                          ;SET UP TO READ MEMORY
U 110A, 3022,15 ;14649          MOV MEM.DATA TO WR[0]           ;GET DATA FROM MEMORY
U 110B, 0869,3C ;14650          JSR [CHECK.RESULT]              ;CHECK RESULT FOR 80000000(H)
U 110C, 8910,64 ;14651          JMP [LOOP.T1E.4]                ;ERROR LOOP IF ENABLED
                ;14652
U 110D, 5B00,1D ;14653          SKIP.IF[MEM.REF.OK]             ;SKIP NEXT INSTRUCTION IF NO ERROR SUM
U 110E, 8911,44 ;14654          JMP [TEST.T1E.5]                ;ELSE GO CHECK CSR FOR RDS
                ;14655  TEST.T1E.4.INC:
U 110F, 6C13,95 ;14656          ADD WR[3] TO LS[T9]             ;INCREMENT CURRENT ADDRESS BY 4
U 1110, 6C89,95 ;14657          ADD WR[3] TO LS[ADDRESS.DATA]   ;ALSO INCREMENT ADDRESS TO PRINT
                ;14658          CMP WR[2] WITH LS[T9],          ;COMPARE NEW ADDRESS WITH LAST ADDRESS + 1
U 1111, CF13,35 ;14659           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1112, 8910,61 ;14660          JMP.IF[NEQ] TO [LOOP.T1E.4]     ;REPEAT UNTIL ALL ADDRESSES CHECKED
U 1113, 0911,F4 ;14661          JMP [END.T1E]                   ;DONE.  GO TO END
                ;14662
                ;14663  TEST.T1E.5:
U 1114, FF82,15 ;14664          INC LS[ERROR.NUMBER]            ;ERROR 5
U 1115, 5F7E,15 ;14665          MCOM LS[RDS] TO WR[0]           ;CLEAR RDS (BIT 31) IN WR
U 1116, 3E8A,15 ;14666          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK RDS BIT ONLY IN CSR 1
U 1117, 2F82,95 ;14667          CLR WR[1]                       ;EXPECTED DATA (RDS CLEAR)
U 1118, 9D45,75 ;14668          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;14669                                          ;SET UP TO READ CSR 1
U 1119, 3022,15 ;14670          MOV MEM.DATA TO WR[0]           ;GET CSR 1 CONTENTS
U 111A, 0869,3C ;14671          JSR [CHECK.RESULT]              ;SEE IF RDS IS SET
U 111B, 0910,44 ;14672          JMP [LOOP.T1E.5]                ;ERROR LOOP IF ENABLED
U 111C, FC82,15 ;14673          DEC LS[ERROR.NUMBER]            ;RESTORE ERROR NUMBER TO ERROR 3
U 111D, E58A,15 ;14674          CLR LS[ERROR.MASK]              ;RESTORE MASK TO CHECK ALL BITS
U 111E, 8910,F4 ;14675          JMP [TEST.T1E.4.INC]            ;GO TO GENERATE NEXT ADDRESS
                ;14676
                ;14677  END.T1E:
```

```
                ;14678  .PAGE   '' TEST 1F Data Path Test (M8728  ARRAY)''
                ;14679  ;++
                ;14680  ;
                ;14681  ; Test Description:
                ;14682  ;
                ;14683  ;       This test checks the data path on the array boards for shorts between
                ;14684  ;       bits.  This test will check one address on each 256KB block to assure
                ;14685  ;       that all array cards are tested.  The test will write and read a shift-
                ;14686  ;       ing 1's through a field of zeros pattern across all 32 bits.
                ;14687  ;
                ;14688  ; Assumptions:
                ;14689  ;
                ;14690  ;       It is assumed that all previous tests have run successfully.
                ;14691  ;
                ;14692  ; Test Steps:
                ;14693  ;
                ;14694  ;       1)  Set up error mask, error number, and module number in LS (for
                ;14695  ;           error printouts) and clear previous error number in LS.
                ;14696  ;       2)  Clear CSR 1 control bits, load a temporary LS location with 0's
                ;14697  ;           (will be used as the address) and load LS MM.LASTADDR into WR 2.
                ;14698  ;       3)  Check that WR 2 (last memory address+1) is non-zero.  Size memory
                ;14699  ;           and print size if memory not sized previously.
                ;14700  ;       4)  Clear the OS reg (used to index) and load WR 1 with the first data
                ;14701  ;           pattern.
                ;14702  ;       5)  Execute a MEM.REQ with MF=WRITE.P and DT=longword.  Use the temp-
                ;14703  ;           orary LS location as the address.
                ;14704  ;       6)  Execute a WRITE.MEM LS instruction using index mode.
                ;14705  ;       7)  Execute a MEM.REQ with MF=READ.P and DT=longword.  Use the temp-
                ;14706  ;           orary LS location as the address.
                ;14707  ;       8)  Execute MOV MEM.DATA TO WR[0] instruction and check result.
                ;14708  ;       9)  Increment OS and repeat steps 3 through 7 until all 32 bits have
                ;14709  ;           been tested.
                ;14710  ;       10) Increment the temporary LS location used for the memory address
                ;14711  ;           by 40000(H) (256KB) and compare with LS MM.LASTADDR+1.  If these
                ;14712  ;           are equal, we are done.  Else repeat steps 3 through 7 with a
                ;14713  ;           new address.
                ;14714  ;
                ;14715  ; Errors:
                ;14716  ;
                ;14717  ;       Error 1 - ARRAY failed shifting pattern test. Address under OTHER DATA.
                ;14718  ;
                ;14719  ; Debug:
                ;14720  ;
                ;14721  ;       NOTE:  If the MCT is single stepped, refresh on the ARRAY module will
                ;14722  ;       be lost.  Thus the array data will be lost.
                ;14723  ;
                ;14724  ;       Error 1 - Check for a short between the failing bits on the ARRAY board
                ;14725  ;       using the address under OTHER DATA to determine the failing module.
                ;14726  ;
                ;14727  ;--
                ;14728
                ;14729  T.1F:
                ;14730
U 111F, B65E,15 ;14731          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;14732                                          ; STATUS WORD
```

```
U 1120, 3E80,15 ;14733          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;14734                                         ; BIT 15 INDICATES START OF TEST TO
                ;14735                                         ; CONSOLE
U 1121, 10E0,15 ;14736          MISC [SET.CP.ATTN]             ;ISSUE CPU ATTN TO CONSOLE
                ;14737  WAIT.T1F.0:
U 1122, 8912,24 ;14738          JMP [WAIT.T1F.0]               ;LOOP HERE WAITING FOR CONSOLE TO
                ;14739                                         ; RESPOND
                ;14740
U 1123, 0A1A,9C ;14741          JSR [SETUP]                    ;SET UP MASKS, MODULE CODE ETC. AND
                ;14742                                         ; CLEAR MCT CSR 1
U 1124, 3648,15 ;14743          MOV LS[ARRAY] TO WR[0]         ;SET BIT FOR ARRAY BOARD
U 1125, 3E8C,15 ;14744          MOV WR[0] TO LS[MODULE.NUM]    ;STORE MODULE CODE IN LS TO INDICATE
                ;14745                                         ; ARRAY BOARD
                ;14746
                ;14747          MOV LS[MM.LASTADDR+1] TO WR[2], ;GET LAST ADDRESS + 1 FROM LS
U 1126, 3625,35 ;14748           DT(LONG)&SET.ALU.CC           ; AND SET CONDITION CODES
U 1127, 0912,91 ;14749          JMP.IF[NEQ] TO [T1F.CONTINUE]  ;CONTINUE TESTING IF LAST ADDRESS+1 IS
                ;14750                                         ; NON-ZERO
U 1128, 8A19,4C ;14751          JSR [SIZE.MEMORY]              ;SIZE MEMORY IF NOT DONE PREVIOUSLY AND
                ;14752                                         ; PRINT RESULT
                ;14753
                ;14754  T1F.CONTINUE:
U 1129, B670,15 ;14755          MOV LS[OTHER.DATA] TO WR[0]    ;SET OTHER DATA PRINTOUT BIT
U 112A, 3E80,15 ;14756          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL WORD TO PRINT ADDRESS
                ;14757                                         ; UNDER 'OTHER' IF ERROR
U 112B, 6512,15 ;14758          CLR LS[T9]                     ;ADDRESS TO ACCESS IN MAIN MEMORY
U 112C, 6588,15 ;14759          CLR LS[ADDRESS.DATA]           ;ALSO CLEAR ADDRESS TO PRINT
                ;14760  REPEAT.T1F.1:
U 112D, E5F8,15 ;14761          CLR LS[OS]                     ;CLEAR INDEX
                ;14762  LOOP.T1F.1:
U 112E, B6F6,95 ;14763          MOV LS[SHIFT.OS(4-0)] TO WR[1]  ;EXPECTED DATA (SHIFT PATTERN)
U 112F, 9912,75 ;14764          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;14765                                         ;SET UP TO WRITE MAIN MEMORY
U 1130, B2F6,15 ;14766          WRITE.MEM LS[SHIFT.OS(4-0)]    ;WRITE SHIFT PATTERN IN MEMORY
U 1131, 1813,F5 ;14767          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;14768                                         ;SET UP TO READ RESULT
U 1132, 3022,15 ;14769          MOV MEM.DATA TO WR[0]          ;GET RESULT
U 1133, 0869,3C ;14770          JSR [CHECK.RESULT]             ;CHECK FOR AN ERROR
U 1134, 8912,E4 ;14771          JMP [LOOP.T1F.1]               ;ERROR LOOP IF ENABLED
                ;14772
U 1135, 7FF8,15 ;14773          INC LS[OS]                     ;INCREMENT INDEX FOR NEXT PATTERN
U 1136, B6F8,15 ;14774          MOV LS[OS] TO WR[0]            ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                ;14775          BIT LS[BIT5] WITH WR[0],       ;SEE IF OS HAS INCREMENTED TO 20(H)
U 1137, 594A,35 ;14776           DT(LONG)&SET.ALU.CC           ; AND SET CONDITION CODES
U 1138, 0912,E9 ;14777          JMP.IF[BITS.CLR] TO [LOOP.T1F.1] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                ;14778
U 1139, E5F8,15 ;14779          CLR LS[OS]                     ;CLEAR INDEX
U 113A, B664,15 ;14780          MOV LS[#40000] TO WR[0]        ;GET 256KB BLOCK INCREMENT
U 113B, 6C12,15 ;14781          ADD WR[0] TO LS[T9]            ;ADD TO CURRENT ADDRESS
U 113C, 6C88,15 ;14782          ADD WR[0] TO LS[ADDRESS.DATA]  ;SAME WITH ADDRESS TO PRINT
                ;14783          CMP WR[2] WITH LS[T9],         ;SEE IF ANY MORE BLOCKS (COMPARE TO
                ;14784                                         ; LAST ADDRESS + 1)
U 113D, CF13,35 ;14785           DT(LONG)&SET.ALU.CC           ; AND SET CONDITION CODES
U 113E, 8912,D1 ;14786          JMP.IF[NEQ] TO [REPEAT.T1F.1]  ;REPEAT IF NOT, ELSE DONE
                ;14787  END.T1F:
```

```
;14788  .PAGE   "  TEST 20 Refresh Test (M8728  ARRAY, M8394  WCS Module)"
;14789  ;++
;14790  ;
;14791  ; Test Description:
;14792  ;
;14793  ;       This test is similiar to the all 1's and all 0's test run previously.
;14794  ;       The difference is that all available memory is written with a back-
;14795  ;       ground of all 1's or all 0's and read back after a delay exceeding the
;14796  ;       minimum refresh level required to sustain the data in the RAM chip.  If
;14797  ;       the refresh logic fails, the data will decay and an error will be re-
;14798  ;       ported.  The test is run twice, once with all ones as the background
;14799  ;       data and once with all zeros.  This will assure that a decay to either
;14800  ;       state will be detected (most of the time).
;14801  ;
;14802  ; Assumptions:
;14803  ;
;14804  ;       It is assumed that all previous tests have run successfully.  This test
;14805  ;       will only detect a total lack of refresh and is reasonably accurate
;14806  ;       only at temperatures of 50 degrees centigrade or above.
;14807  ;
;14808  ; Test Steps:
;14809  ;
;14810  ;       1)  Set up error mask, error number, and module number in LS (for
;14811  ;           error printouts) and clear previous error number in LS.
;14812  ;       2)  Clear CSR 1 control bits, load a temporary LS location with 0's
;14813  ;           (will be used as the address) and load LS MM.LASTADDR into WR 2.
;14814  ;       3)  Check that WR 2 (last memory address+1) is non-zero.  Size memory
;14815  ;           and report result if not sized previously.
;14816  ;       4)  Execute a MEM.REQ with MF=WRITE.P and DT=LONGWORD.  Use the temp-
;14817  ;           orary LS location as the address in main memory.
;14818  ;       5)  Execute a WRITE.MEM LS with data of 1's.
;14819  ;       6)  Increment the LS location by 4 and repeat steps 4 through 6 until
;14820  ;           the last memory address is exceeded.
;14821  ;       7)  Delay with a micro-code loop for about 2 seconds.
;14822  ;       8)  Read each location and check the result.
;14823  ;       9)  Repeat steps 4 through 8 with data of 0's.
;14824  ;
;14825  ; Errors:
;14826  ;
;14827  ;       Error 1 - Refresh logic failure (a failure at only a couple addresses
;14828  ;                 may indicate a bad RAM chip).  Address under OTHER data.
;14829  ;
;14830  ; Debug:
;14831  ;
;14832  ;       NOTE:  If the MCT is single stepped, refresh on the ARRAY module will
;14833  ;       be lost.  Thus the array data will be lost.
;14834  ;
;14835  ;       Error 1 - This error indicates a problem with the refresh logic on the
;14836  ;       WCS module or the ARRAY module.  In a small number of cases, it may in-
;14837  ;       dicate a problem in a RAM chip itself due to cell leakage.  The first
;14838  ;       signals to check are ARRAY REF CYC L and ARRAY RAS TIM L out of the WCS
;14839  ;       module.  This can be checked while the machine is idleing as refresh
;14840  ;       always runs.  If ARRAY REF CYC L is pulsing, check the address lines
;14841  ;       (BUS ARRAY A0 H through BUS ARRAY A7 H) to make sure that they are in-
;14842  ;       crementing properly.  Each of these lines should be toggleing at half
```

```
                ;14843  ;       the rate of the previous one when starting at address line 0 and check-
                ;14844  ;       ing through address line 7.  If all these signals are working, the
                ;14845  ;       problem lies on the ARRAY module.
                ;14846  ;         If ARRAY REF CYC L or ARRAY RAS TIM L are not pulsing, check the in-
                ;14847  ;       puts to the REFR CNTRL PAL for a high on both ALLOW REF H and INH REF
                ;14848  ;       CYC L.  ALLOW REF H is a control store bit from the memory controler.
                ;14849  ;       If these are OK check that MAIN MEM REFR L and 90 NS CP H are pulsing.
                ;14850  ;       If these are OK, the PAL is probably bad.
                ;14851  ;         If there is a problem with BUS ARRAY A0 H through BUS ARRAY A7 H,
                ;14852  ;       check the inputs to the QUAD DRIVER and the BIN CTR.
                ;14853  ;
                ;14854  ;--
                ;14855
                ;14856  T.20:
                ;14857
U 113F, B65E,15 ;14858          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;14859                                          ; STATUS WORD
U 1140, 3E80,15 ;14860          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;14861                                          ; BIT 15 INDICATES START OF TEST TO
                ;14862                                          ; CONSOLE
U 1141, 10E0,15 ;14863          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;14864  WAIT.T20.0:
U 1142, 8914,24 ;14865          JMP [WAIT.T20.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;14866                                          ; RESPOND
                ;14867
U 1143, 0A1A,9C ;14868          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                ;14869                                          ; CLEAR MCT CSR 1
U 1144, B640,15 ;14870          MOV LS[WCS] TO WR[0]            ;SET UP MODULE CODE. BIT 1 SET IN WR
U 1145, 4748,15 ;14871          BIS LS[ARRAY] TO WR[0]          ;SET BIT FOR ARRAY BOARD
U 1146, 3E8C,15 ;14872          MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                ;14873                                          ; WCS OR ARRAY BOARD
                ;14874
                ;14875          MOV LS[MM.LASTADDR+1] TO WR[2], ;GET LAST ADDRESS + 1 FROM LS
U 1147, 3625,35 ;14876           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1148, 0914,A1 ;14877          JMP.IF[NEQ] TO [T20.CONTINUE]   ;CONTINUE TESTING IF LAST ADDRESS+1 IS
                ;14878                                          ; NON-ZERO
U 1149, 8A19,4C ;14879          JSR [SIZE.MEMORY]               ;SIZE MEMORY IF NOT DONE PREVIOUSLY AND
                ;14880                                          ; REPORT RESULT
                ;14881
                ;14882  T20.CONTINUE:
U 114A, 3645,95 ;14883          MOV LS[#4] TO WR[3]             ;VALUE TO INCREMENT ADDRESS BY
U 114B, 6512,15 ;14884          CLR LS[T9]                      ;ADDRESS TO ACCESS IN MAIN MEMORY
                ;14885  BACK.T20.1:
U 114C, 9912,75 ;14886          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;14887                                          ;SET UP TO WRITE BACKGROUND OF 1'S
U 114D, 329E,15 ;14888          WRITE.MEM LS[ONES]              ;WRITE ONES
U 114E, 6C13,95 ;14889          ADD WR[3] TO LS[T9]             ;INCREMENT ADDRESS
                ;14890          CMP WR[2] WITH LS[T9],          ;SEE IF DONE ALL MEMORY
U 114F, CF13,35 ;14891           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1150, 0914,C1 ;14892          JMP.IF[NEQ] TO [BACK.T20.1]     ;REPEAT UNTIL ALL MEMORY WRITTEN
                ;14893
U 1151, 6512,15 ;14894          CLR LS[T9]                      ;ADDRESS TO ACCESS IN MAIN MEMORY
U 1152, 9912,75 ;14895          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;14896                                          ;SET UP TO WRITE 1'S IN LOCATION 0 AGAIN
U 1153, 329E,15 ;14897          WRITE.MEM LS[ONES]              ;TO SET UP ADDRESS LINES TO A LOW
```

```
                    ;14898
                    ;14899          JSR [DELAY.T20]                 ;DO 5 SECOND DELAY
                    ;14900
U 1154, 0917,DC     ;14901          MOV LS[OTHER.DATA] TO WR[0]      ;SET OTHER DATA PRINTOUT BIT
U 1155, B670,15     ;14902          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL WORD TO PRINT ADDRESS
U 1156, 3E80,15     ;14903                                          ; UNDER 'OTHER' IF ERROR
                    ;14904          CLR LS[T9]                      ;START AT ADDRESS 0
U 1157, 6512,15     ;14905          CLR LS[ADDRESS.DATA]            ;ALSO CLEAR ADDRESS TO PRINT
U 1158, 6588,15     ;14906  LOOP.T20.1:
U 1159, 369E,95     ;14907          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 115A, 1813,F5     ;14908          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                    ;14909                                          ;SET UP TO READ BACK DATA
U 115B, 3022,15     ;14910          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 115C, 0869,3C     ;14911          JSR [CHECK.RESULT]              ;CHECK FOR 1'S
U 115D, 8915,94     ;14912          JMP [LOOP.T20.1]                ;ERROR LOOP IF ENABLED
                    ;14913
U 115E, 6C13,95     ;14914          ADD WR[3] TO LS[T9]              ;INCREMENT ADDRESS
U 115F, 6C89,95     ;14915          ADD WR[3] TO LS[ADDRESS.DATA]    ;ALSO INCREMENT ADDRESS TO PRINT
                    ;14916          CMP WR[2] WITH LS[T9],          ;SEE IF DONE ALL MEMORY
U 1160, CF13,35     ;14917           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 1161, 8915,91     ;14918          JMP.IF[NEQ] TO [LOOP.T20.1]     ;REPEAT UNTIL ALL MEMORY IS READ
                    ;14919
U 1162, E580,15     ;14920          CLR LS[CONTROL.STATUS]          ;CLEAR CONTROL AND STATUS IN CASE
                    ;14921                                          ; THERE WAS A PREVIOUS ERROR
U 1163, 10E0,15     ;14922          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE (TO PREVENT
                    ;14923                                          ; TIMEOUT ON LARGER MEMORY SYSTEMS)
                    ;14924  WAIT.T20.1:
U 1164, 0916,44     ;14925          JMP [WAIT.T20.1]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;14926                                          ; RESPOND
U 1165, 6512,15     ;14927          CLR LS[T9]                      ;ADDRESS TO ACCESS IN MAIN MEMORY
                    ;14928  BACK.T20.1A:
U 1166, 9912,75     ;14929          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                    ;14930                                          ;SET UP TO WRITE BACKGROUND OF 0'S
U 1167, B29C,15     ;14931          WRITE.MEM LS[ZERO]              ;WRITE ZEROS
U 1168, 6C13,95     ;14932          ADD WR[3] TO LS[T9]              ;INCREMENT ADDRESS
                    ;14933          CMP WR[2] WITH LS[T9],          ;SEE IF DONE ALL MEMORY
U 1169, CF13,35     ;14934           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 116A, 8916,61     ;14935          JMP.IF[NEQ] TO [BACK.T20.1A]    ;REPEAT UNTIL ALL MEMORY WRITTEN
                    ;14936
U 116B, 6512,15     ;14937          CLR LS[T9]                      ;ADDRESS TO ACCESS IN MAIN MEMORY
U 116C, 9912,75     ;14938          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                    ;14939                                          ;SET UP TO WRITE 0'S IN LOCATION 0 AGAIN
U 116D, B29C,15     ;14940          WRITE.MEM LS[ZERO]              ;TO SET UP ADDRESS LINES TO A LOW
                    ;14941
U 116E, 0917,DC     ;14942          JSR [DELAY.T20]                 ;DELAY FOR 5 SECONDS
                    ;14943
U 116F, B670,15     ;14944          MOV LS[OTHER.DATA] TO WR[0]      ;SET OTHER DATA PRINTOUT BIT
U 1170, 3E80,15     ;14945          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL WORD TO PRINT ADDRESS
                    ;14946                                          ; UNDER 'OTHER' IF ERROR
U 1171, 6512,15     ;14947          CLR LS[T9]                      ;START AT ADDRESS 0
U 1172, 6588,15     ;14948          CLR LS[ADDRESS.DATA]            ;ALSO CLEAR ADDRESS TO PRINT
                    ;14949  LOOP.T20.1A:
U 1173, 2F82,95     ;14950          CLR WR[1]                       ;EXPECTED DATA
U 1174, 1813,F5     ;14951          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                    ;14952                                          ;SET UP TO READ BACK DATA
```

```
U 1175, 3022,15 ;14953          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 1176, 0869,3C ;14954          JSR [CHECK.RESULT]               ;CHECK FOR 1'S
U 1177, 0917,34 ;14955          JMP [LOOP.T20.1A]                ;ERROR LOOP IF ENABLED
                ;14956
U 1178, 6C13,95 ;14957          ADD WR[3] TO LS[T9]              ;INCREMENT ADDRESS
U 1179, 6C89,95 ;14958          ADD WR[3] TO LS[ADDRESS.DATA]    ;ALSO INCREMENT ADDRESS TO PRINT
                ;14959          CMP WR[2] WITH LS[T9],           ;SEE IF DONE ALL MEMORY
U 117A, CF13,35 ;14960           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 117B, 0917,31 ;14961          JMP.IF[NEQ] TO [LOOP.T20.1A]     ;REPEAT UNTIL ALL MEMORY IS READ
U 117C, 0918,64 ;14962          JMP [END.T20]                    ;DONE
                ;14963
                ;14964  ;
                ;14965  ;DELAY SUBROUTINE
                ;14966  ;
                ;14967
                ;14968  DELAY.T20:
U 117D, B66E,15 ;14969          MOV LS[BIT23] TO WR[0]           ;COUNTER (800000(H))
U 117E, E580,15 ;14970          CLR LS[CONTROL.STATUS]           ;CLEAR CONTROL AND STATUS WORD
U 117F, 10E0,15 ;14971          MISC [SET.CP.ATTN]               ;ISSUE CPU ATTN TO CONSOLE (TO PREVENT
                ;14972                                           ; TIMEOUT ON LARGER MEMORY SYSTEMS)
                ;14973  DELAY.T20.0:
                ;14974          DEC WR[0],                       ;DECREMENT COUNTER
U 1180, A100,35 ;14975           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 1181, 0918,01 ;14976          JMP.IF[NEQ] TO [DELAY.T20.0]     ;REPEAT UNTIL DELAY OF 5 SECONDS
U 1182, E580,15 ;14977          CLR LS[CONTROL.STATUS]           ;CLEAR CONTROL AND STATUS WORD
U 1183, 10E0,15 ;14978          MISC [SET.CP.ATTN]               ;ISSUE CPU ATTN TO CONSOLE (TO PREVENT
                ;14979                                           ; TIMEOUT ON LARGER MEMORY SYSTEMS)
                ;14980  DELAY.T20.1:
U 1184, 8918,44 ;14981          JMP [DELAY.T20.1]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;14982                                           ; RESPOND
U 1185, 5B00,14 ;14983          RETURN                           ;DONE
                ;14984
                ;14985
                ;14986  END.T20:
```

```
;14987  .PAGE   "  TEST 21 Main Memory March Test (M8391 MCT or M8728 ARRAY Module)"
;14988  ;++
;14989  ;
;14990  ; Test Description:
;14991  ;
;14992  ;       This test runs a march test throughout all available memory.  It first
;14993  ;       checks to see that the memory has been sized and initialized by check-
;14994  ;       ing LS location MM.LASTADDR+1.  If this value is 0, the memory is sized
;14995  ;       and the result is reported.  After this, the memory is written with a
;14996  ;       background of zeros.  Starting at location 0, each longword memory loc-
;14997  ;       ation is read for 0's, written to ones, and read for 1's.  If the data
;14998  ;       is OK, the error sum bit is checked.  If error sum is asserted, CSR 1
;14999  ;       is read and the CRD bit is checked (single bit error).  If this is the
;15000  ;       only error bit set in the CSR, the resulting syndrome bits are checked
;15001  ;       to assure that the ckbts are not causing the failure.  A count is in-
;15002  ;       cremented to indicate the number of single bit errors detected.  If any
;15003  ;       other error bits in the CSR are set, an error is reported (CRD is the
;15004  ;       only error that should occur if the data came back correctly).  A sep-
;15005  ;       erate single error count will be kept for each 256 KB of memory.  With
;15006  ;       the 16K RAM array card, this represents 1 array module.
;15007  ;         The address is incremented and the read/write/read sequence is re-
;15008  ;       peated until the top of memory is reached.  Then the test is repeated
;15009  ;       starting at the top of memory and decrementing the address until loc-
;15010  ;       ation 0 is checked.  The descending march pattern reads for 1's, writes
;15011  ;       0's and reads 0's.  Only RDS errors (i.e. uncorrectable errors) are re-
;15012  ;       ported.  Single bit errors are not counted during the descending pass.
;15013  ;         After the march test with all ones and all zeros, another march is
;15014  ;       run with data patterns of 0's and 80000000(H).  The pattern with bit 31
;15015  ;       set complements the ckbts and thus provides a march test on them.  The
;15016  ;       same read/write/read pattern is used as before, except only single bit
;15017  ;       ckbt failures will be counted (single bit data failures were counted in
;15018  ;       the first march test).  The single bit data and single bit ckbt counts
;15019  ;       will be added together to form the total CRD failures per 256KB block.
;15020  ;
;15021  ; Assumptions:
;15022  ;
;15023  ;       It is assumed that all previous tests have run successfully.  Errors
;15024  ;       that occur in this test are most likely address related (except single
;15025  ;       bit errors that do not generate a normal error report).
;15026  ;
;15027  ; Test Steps:
;15028  ;
;15029  ;       1)  Set up error mask, error number, and module number in LS (for
;15030  ;           error printouts) and clear previous error number in LS.
;15031  ;       2)  Load WR 2 with the memory size (LS location MM.LASTADDR+1) and
;15032  ;           check that size is not 0.  If size is 0, resize memory, report
;15033  ;           result and continue.
;15034  ;       3)  Clear CSR 1 and clear a temporary LS location (for the address).
;15035  ;       4)  Write a background of zeros throughout main memory.  Clear the
;15036  ;           temporary LS location after background is written.
;15037  ;       5)  Load WR 1 with 0's (expected data) and execute MEM.REQ with MF=
;15038  ;           READ.P and DT=longword.  Use the temporary LS location as the
;15039  ;           address.
;15040  ;       6)  Execute MOV MEM.DATA to WR[0] and check result.  If error, report
;15041  ;           and go to step 9.
```

```
;15042  ;       7)  Check for an ERROR SUM.  If no ERROR SUM, go to step 9, else read
;15043  ;           CSR 1 and check that only CRD is set.
;15044  ;       8)  If CRD, read CSR 0 (syndrome bits) and check that error is in data
;15045  ;           portion rather than the ckbits.  Increment single error count in LS
;15046  ;           if data portion had the error.  If APT or single error report is
;15047  ;           enabled, read location with ECC disabled and report error.
;15048  ;       9)  Load WR 1 with 1's and execute MEM.REQ with MF=WRITE.P and DT=
;15049  ;           longword.  Execute WRITE.MEM LS with ones in LS.
;15050  ;       10) Repeat steps 6 through 8 checking for data of 1's.
;15051  ;       11) Increment LS location containing the address by 4.  Compare with
;15052  ;           LS MM.LASTADDR+4.  If not equal go to step 5, else go to next step.
;15053  ;       12) Repeat steps 5 through 10 but skip steps 7 and 8.  Use a complement
;15054  ;           data pattern and descending addresses.
;15055  ;       13) Repeat steps 5 through 11.  In step 8 check that error is in ckbts
;15056  ;           and in step 9, use a data pattern of 80000000(H).  If APT or single
;15057  ;           error report is enabled, print out complemented syndrome bits
;15058  ;           that cause an error.
;15059  ;
;15060  ; Errors:
;15061  ;
;15062  ;       NOTE:     To print out single bit errors as normal errors, type SE SE
;15063  ;                 (set single error report) on the terminal and run this test.
;15064  ;                 Single bit errors will be detected and the address and failing
;15065  ;                 data will be printed as error 1 or 4.
;15066  ;
;15067  ;       Error 1 - March test failed on ascending addresses with data of all 0's
;15068  ;                 and all 1's.  If APT or single errors report is enabled,
;15069  ;                 single bit errors will also be reported here.
;15070  ;       Error 2 - Error bit other than CRD set in CSR even though data was cor-
;15071  ;                 rect.
;15072  ;       Error 3 - March test failed on descending addresses with data of all
;15073  ;                 0's and all 1's.
;15074  ;       Error 4 - March test failed on ascending addresses with data of all
;15075  ;                 0's and 80000000(H).  If APT or single error report set, the
;15076  ;                 syndrome bits received will be reported here (SYNDROMES COME
;15077  ;                 BACK COMPLEMENTED).
;15078  ;       Error 5 - Error bit other than CRD set in CSR even though data was cor-
;15079  ;                 rect.
;15080  ;
;15081  ; Debug:
;15082  ;
;15083  ;       NOTE:  If the MCT is single stepped, refresh on the ARRAY module will
;15084  ;       be lost.  Thus the array data will be lost.
;15085  ;
;15086  ;       Two methods of debug will be explained.  If the MCT module is in a
;15087  ;       test station then the MCT can be single stepped making debug easier
;15088  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;15089  ;       method will be explained also.  When single stepping the memory con-
;15090  ;       troller will help in debug, this section will explicitly suggest it.
;15091  ;
;15092  ;       NOTE:**Error 1 could be caused by any array holding an address bit
;15093  ;       low.  Removing all array boards and replacing with a known good array
;15094  ;       will help to isolate this type of problem.
;15095  ;
;15096  ;       Error 1 - This error can be caused by an address logic failure on the
```

```
;15097  ;       MCT module or the ARRAY module, or an internal RAM chip failure.  Add-
;15098  ;       ress logic failures will occur on addresses that have a new bit assert-
;15099  ;       ed (e.g. at address 10(H) where bit 4 is first asserted) and all bits
;15100  ;       will fail.  RAM chip failures will show up at any address and have few
;15101  ;       bits (probably 2) failing.
;15102  ;         If the errors look like address errors (all bits failing), then the
;15103  ;       first signals to check would be the address lines BUS ARRAY A0 H
;15104  ;       through BUS ARRAY A6 H coming out of the address muxes on the MCT mod-
;15105  ;       ule.  With loop on error enabled, look at the BUS ARRAY Ax H lines.
;15106  ;       The signals should show the address being accessed (printed under OTHER
;15107  ;       DATA) with the lower 7 bits coming out when COLAD H is low and the up-
;15108  ;       per 7 bits coming out when COLAD H is high.  On 64K RAM modules, the
;15109  ;       signal A7 H should also be checked.  This should correspond to bit
;15110  ;       16 and 17 of the address.  The scope can be synced on the signal which
;15111  ;       goes to pin 1 of the delay line, produced by RAS EN H when REF IN PROG
;15112  ;       L is high.  Syncing on this signal will eliminate triggering when a re-
;15113  ;       fresh occurs (which turns off the enable on the muxes) and also signals
;15114  ;       the beginning of the access to memory.
;15115  ;         If the address lines are in error, check the inputs to the muxes.  If
;15116  ;       these are OK, the MUX is probably bad.
;15117  ;         If these address lines are OK, the BUS ARRAY B SEL 0 H and BUS ARRAY
;15118  ;       SEL 1 H should be checked.  These select the correct bank of chips on
;15119  ;       the array card.  If the error occurred on a bank boundary (for the 16K
;15120  ;       RAM array this is when address bits 16 or 17 change, on a 64K RAM array
;15121  ;       this occurs when address bits 18 or 19 change), this area is suspect.
;15122  ;       Use the same sync point as before and check that the BUS ARRAY B SEL x
;15123  ;       H lines correspond with the address that failed.
;15124  ;         If the select lines are in error, check the inputs, including the SEL
;15125  ;       input for the proper value.  If the inputs are OK, the MUX is probably
;15126  ;       bad.
;15127  ;         If the select lines are OK, the only likely are left on the MCT mod-
;15128  ;       ule is the MEM SEL lines.  This is possible if the error occurred on
;15129  ;       an address that crossed an array boundary.  Check the MEM SEL x L sig-
;15130  ;       nal for a low on the line that corresponds to the array that should
;15131  ;       have been selected.  If this is incorrect, the fingerprint signals
;15132  ;       should be checked for the correct value for the array modules used.
;15133  ;       If the fingerprint signal is OK, the DECODER PAL which produces the
;15134  ;       MEM SEL x L signal needed is probably bad.
;15135  ;         If all the addressing logic on the MCT board appears to be working,
;15136  ;       then the problem probably lies on the array baord itself.
;15137  ;
;15138  ;       Error 2 - Use the EX MC 01 command to determine which error bit is
;15139  ;       failing and trace that bit back.  A double error in the CKBTs could
;15140  ;       cause this problem (RDS will be set in CSR 1 in this case).
;15141  ;
;15142  ;       Error 3 - This error most likely indicates a faulty RAM chip.  The chip
;15143  ;       can be located by observing the bits that are failing and the address.
;15144  ;
;15145  ;       Error 4 - This error should only occur on addresses that failed in er-
;15146  ;       ror 1 above unless this is reporting the failing CKBTS.  If the error
;15147  ;       mask is FFFFFF80, this error is reporting the resulting syndrome bits
;15148  ;       after a single bit error.
;15149  ;
;15150  ;       Error 5 - Use the EX MC 01 command to determine which error bit is
;15151  ;       failing and trace that bit back.  If the RDS bit is setting, it could
```

```
                ;15152  ;       indicate an error in the CKBT portion of the RAM.  This could be caused
                ;15153  ;       by an address line break or a faulty RAM chip.
                ;15154  ;
                ;15155  ;--
                ;15156
                ;15157  T.21:
                ;15158
U 1186, B65E,15 ;15159          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;15160                                          ; STATUS WORD
U 1187, 3E80,15 ;15161          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;15162                                          ; BIT 15 INDICATES START OF TEST TO
                ;15163                                          ; CONSOLE
U 1188, 10E0,15 ;15164          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;15165  WAIT.T21.0:
U 1189, 0918,94 ;15166          JMP [WAIT.T21.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;15167                                          ; RESPOND
                ;15168
U 118A, 0A1A,9C ;15169          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                ;15170                                          ; CLEAR MCT CSR 1
U 118B, 4748,15 ;15171          BIS LS[ARRAY] TO WR[0]          ;SET BIT FOR ARRAY BOARD
U 118C, 3E8C,15 ;15172          MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                ;15173                                          ; MCT OR ARRAY BOARD
                ;15174
                ;15175          MOV LS[MM.LASTADDR+1] TO WR[2], ;GET LAST ADDRESS + 1 FROM LS
U 118D, 3625,35 ;15176           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 118E, 8919,01 ;15177          JMP.IF[NEQ] TO [T21.CONTINUE]   ;CONTINUE TESTING IF LAST ADDRESS+1 IS
                ;15178                                          ; NON-ZERO
U 118F, 8A19,4C ;15179          JSR [SIZE.MEMORY]               ;SIZE MEMORY IF NOT DONE PREVIOUSLY AND
                ;15180                                          ; REPORT RESULT
                ;15181  T21.CONTINUE:
U 1190, 3645,95 ;15182          MOV LS[#4] TO WR[3]             ;INCREMENT BY 4
U 1191, 6512,15 ;15183          CLR LS[T9]                      ;ADDRESS TO ACCESS IN MAIN MEMORY
                ;15184  BACK.T21.1:
U 1192, 9912,75 ;15185          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;15186                                          ;SET UP TO WRITE ALL MEMORY
U 1193, B29C,15 ;15187          WRITE.MEM LS[ZERO]              ;WRITE ZEROS
U 1194, 6C13,95 ;15188          ADD WR[3] TO LS[T9]             ;NEXT ADDRESS
                ;15189          CMP WR[2] WITH LS[T9],          ;DONE ALL ADDRESSES?
U 1195, CF13,35 ;15190           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 1196, 0919,21 ;15191          JMP.IF[NEQ] TO [BACK.T21.1]     ;REPEAT UNTIL ALL ADDRESSES WRITTEN
                ;15192
U 1197, 6514,15 ;15193          CLR LS[T10]                     ;CLEAR SINGLE BIT ERROR COUNT
U 1198, 6512,15 ;15194          CLR LS[T9]                      ;CLEAR STARTING ADDRESS FOR MARCH TEST
U 1199, 651E,15 ;15195          CLR LS[T15]                     ;CLEAR ARRAY NUMBER FOR SINGLE ERROR
                ;15196                                          ; PRINTOUT
U 119A, FF1E,15 ;15197          INC LS[T15]                     ;START WITH ARRAY NUMBER 1
U 119B, 6588,15 ;15198          CLR LS[ADDRESS.DATA]            ;ALSO CLEAR ADDRESS TO PRINT
U 119C, 3665,15 ;15199          MOV LS[#40000] TO WR[2]         ;MARCH ON ONE 256KB BLOCK AT A TIME
                ;15200                                          ; (THIS IS LAST ADDRESS+1 OF FIRST BLOCK)
                ;15201  LOOP.T21.2:
U 119D, E58A,15 ;15202          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
U 119E, B670,15 ;15203          MOV LS[OTHER.DATA] TO WR[0]     ;SET OTHER DATA PRINTOUT BIT
U 119F, 3E80,15 ;15204          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL WORD TO PRINT ADDRESS
                ;15205                                          ; UNDER 'OTHER' IF ERROR
U 11A0, B640,15 ;15206          MOV LS[#1] TO WR[0]             ;ERROR NUMBER
```

```
U 11A1, BE82,15 ;15207          MOV WR[0] TO LS[ERROR.NUMBER]   ;ERROR 1
                ;15208  LOOP.T21.1:
U 11A2, 2F82,95 ;15209          CLR WR[1]                       ;EXPECTED DATA
U 11A3, E516,15 ;15210          CLR LS[T11]                     ;SAVE EXPECTED DATA IN LS FOR CRD REPORT
U 11A4, 1813,F5 ;15211          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;15212                                          ;SET UP TO READ ZEROS
U 11A5, 3022,15 ;15213          MOV MEM.DATA TO WR[0]           ;GET RESULT FROM MEMORY
U 11A6, 0869,3C ;15214          JSR [CHECK.RESULT]              ;CHECK FOR ZEROS
U 11A7, 091A,24 ;15215          JMP [LOOP.T21.1]                ;LOOP ON ERROR IF ENABLED
                ;15216
U 11A8, B680,15 ;15217          MOV LS[CONTROL.STATUS] TO WR[0] ;GET CONTROL AND STATUS WORD
                ;15218          BIT LS[BIT8] WITH WR[0],        ;SEE IF AN ERROR OCCURRED (BIT 8 WILL
                ;15219                                          ; HAVE SET IN THE CHECK.RESULT ROUTINE)
U 11A9, D950,35 ;15220           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 11AA, C550,15 ;15221          BIC LS[BIT8] TO WR[0]           ;CLEAR BIT 8 IF SET
U 11AB, 3E80,15 ;15222          MOV WR[0] TO LS[CONTROL.STATUS] ;RESTORE CONTROL WORD WITH BIT 8 CLEAR
U 11AC, 091B,61 ;15223          JMP.IF[BIT.SET] TO [T21.WRITE.ONES] ;IF ERROR, DON'T CHECK FOR ERROR SUM
                ;15224
U 11AD, 5B00,1D ;15225          SKIP.IF[MEM.REF.OK]             ;ELSE CHECK FOR ERROR SUM
U 11AE, 8922,5C ;15226          JSR [T21.2.ERRORSUM]            ;GO SEE IF CRD OR OTHER ERROR
U 11AF, 091B,64 ;15227          JMP [T21.WRITE.ONES]            ;HERE IF NO LOOP ON ERROR OR NO ERROR
                ;15228                                          ; SUM
U 11B0, 0919,D4 ;15229          JMP [LOOP.T21.2]                ;ERROR LOOP IF ENABLED
                ;15230  LOOP.T21.2A:
U 11B1, E58A,15 ;15231          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
U 11B2, B670,15 ;15232          MOV LS[OTHER.DATA] TO WR[0]     ;SET OTHER DATA PRINTOUT BIT
U 11B3, 3E80,15 ;15233          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL WORD TO PRINT ADDRESS
                ;15234                                          ; UNDER 'OTHER' IF ERROR
U 11B4, B640,15 ;15235          MOV LS[#1] TO WR[0]             ;ERROR NUMBER
U 11B5, BE82,15 ;15236          MOV WR[0] TO LS[ERROR.NUMBER]   ;ERROR 1
                ;15237  T21.WRITE.ONES:
U 11B6, 369E,95 ;15238          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 11B7, BE16,95 ;15239          MOV WR[1] TO LS[T11]            ;SAVE EXPECTED DATA IN LS FOR CRD REPORT
U 11B8, 9912,75 ;15240          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;15241                                          ;SET UP TO WRITE COMPLEMENT DATA IN MEMORY
U 11B9, 329E,15 ;15242          WRITE.MEM LS[ONES]              ;WRITE ONES IN MEMORY
U 11BA, 1813,F5 ;15243          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;15244                                          ;SET UP TO READ NEW DATA
U 11BB, 3022,15 ;15245          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 11BC, 0869,3C ;15246          JSR [CHECK.RESULT]              ;CHECK DATA FOR ONES
U 11BD, 091B,64 ;15247          JMP [T21.WRITE.ONES]            ;ERROR LOOP IF ENABLED
                ;15248
U 11BE, B680,15 ;15249          MOV LS[CONTROL.STATUS] TO WR[0] ;GET CONTROL AND STATUS WORD
                ;15250          BIT LS[BIT8] WITH WR[0],        ;SEE IF AN ERROR OCCURRED (BIT 8 WILL
                ;15251                                          ; HAVE SET IN THE CHECK.RESULT ROUTINE)
U 11BF, D950,35 ;15252           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 11C0, C550,15 ;15253          BIC LS[BIT8] TO WR[0]           ;CLEAR BIT 8 IF SET
U 11C1, 3E80,15 ;15254          MOV WR[0] TO LS[CONTROL.STATUS] ;RESTORE CONTROL WORD WITH BIT 8 CLEAR
U 11C2, 091C,71 ;15255          JMP.IF[BIT.SET] TO [T21.1.INC]  ;IF ERROR, DON'T CHECK FOR ERROR SUM. GO
                ;15256                                          ; DO NEXT ADDRESS
                ;15257
U 11C3, 5B00,1D ;15258          SKIP.IF[MEM.REF.OK]             ;ELSE CHECK FOR ERROR SUM
U 11C4, 8922,5C ;15259          JSR [T21.2.ERRORSUM]            ;GO SEE IF CRD OR OTHER ERROR
U 11C5, 5B00,1E ;15260          SKIP                            ;HERE IF NO LOOP ON ERROR OR NO ERROR
                ;15261                                          ; SUM
```

```
U 11C6, 891B,14 ;15262          JMP [LOOP.T21.2A]                ;ERROR LOOP IF ENABLED ON CSR 1 ERROR
                ;15263
                ;15264  T21.1.INC:
U 11C7, 6C13,95 ;15265          ADD WR[3] TO LS[T9]              ;INCREMENT ADDRESS BY 4
U 11C8, 6C89,95 ;15266          ADD WR[3] TO LS[ADDRESS.DATA]    ;ALSO INCREMENT ADDRESS TO PRINT
                ;15267          CMP WR[2] WITH LS[T9],           ;SEE IF DONE ALL ADDRESSES IN THIS BLOCK
U 11C9, CF13,35 ;15268           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 11CA, 091A,21 ;15269          JMP.IF[NEQ] TO [LOOP.T21.1]      ;REPEAT TEST AT NEW ADDRESS IF NOT DONE
                ;15270                                           ; ELSE START DESCENDING ADDRESSES
                ;15271
U 11CB, 36C0,15 ;15272          MOV LS[#3(H)] TO WR[0]           ;3 IN WR
U 11CC, BE82,15 ;15273          MOV WR[0] TO LS[ERROR.NUMBER]    ;ERROR 3 (DESCENDING ADDRESSES)
U 11CD, 3E13,15 ;15274          MOV WR[2] TO LS[T9]              ;GET TOP ADDRESS+1 OF THIS BLOCK
U 11CE, 3E89,15 ;15275          MOV WR[2] TO LS[ADDRESS.DATA]    ;ADDRESS+1 TO PRINT
U 11CF, 4165,15 ;15276          SUB LS[#40000] FROM WR[2]        ;COMPUTE LOW ADDRESS
                ;15277  REPEAT.T21.3:
U 11D0, EB13,95 ;15278          SUB WR[3] FROM LS[T9]            ;DECREMENT ADDRESS BY 4
U 11D1, EB89,95 ;15279          SUB WR[3] FROM LS[ADDRESS.DATA] ;ALSO DECREMENT ADDRESS TO PRINT
                ;15280  LOOP.T21.3:
U 11D2, 369E,95 ;15281          MOV LS[ONES] TO WR[1]            ;EXPECTED DATA
U 11D3, 1813,F5 ;15282          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;15283                                           ;SET UP TO READ ONES
U 11D4, 3022,15 ;15284          MOV MEM.DATA TO WR[0]            ;GET RESULT FROM MEMORY
U 11D5, 0869,3C ;15285          JSR [CHECK.RESULT]               ;CHECK FOR ONES
U 11D6, 891D,24 ;15286          JMP [LOOP.T21.3]                 ;LOOP ON ERROR IF ENABLED
                ;15287
                ;15288  LOOP.T21.3.COMP:
U 11D7, 2F82,95 ;15289          CLR WR[1]                        ;EXPECTED DATA
U 11D8, 9912,75 ;15290          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;15291                                           ;SET UP TO WRITE COMPLEMENT DATA IN MEMORY
U 11D9, B29C,15 ;15292          WRITE.MEM LS[ZERO]               ;WRITE ZEROS IN MEMORY
U 11DA, 1813,F5 ;15293          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;15294                                           ;SET UP TO READ NEW DATA
U 11DB, 3022,15 ;15295          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 11DC, 0869,3C ;15296          JSR [CHECK.RESULT]               ;CHECK DATA FOR ZEROS
U 11DD, 891D,74 ;15297          JMP [LOOP.T21.3.COMP]            ;ERROR LOOP IF ENABLED
                ;15298
                ;15299          CMP WR[2] WITH LS[T9],           ;SEE IF DONE ALL ADDRESSES IN THIS BLOCK
U 11DE, CF13,35 ;15300           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 11DF, 091D,01 ;15301          JMP.IF[NEQ] TO [REPEAT.T21.3]    ;REPEAT TEST AT NEW ADDRESS IF NOT DONE
                ;15302                                           ; ELSE START WITH NEW DATA PATTERN
                ;15303
U 11E0, 3E13,15 ;15304          MOV WR[2] TO LS[T9]              ;SET UP STARTING ADDRESS
U 11E1, 3E89,15 ;15305          MOV WR[2] TO LS[ADDRESS.DATA]    ;ALSO SET UP ADDRESS TO PRINT
U 11E2, C065,15 ;15306          ADD LS[#40000] TO WR[2]          ;MARCH ON ONE 256KB BLOCK AT A TIME
                ;15307                                           ; (THIS IS LAST ADDRESS+1 OF THIS BLOCK)
                ;15308  LOOP.T21.5:
U 11E3, E58A,15 ;15309          CLR LS[ERROR.MASK]               ;CHECK ALL BITS
U 11E4, B670,15 ;15310          MOV LS[OTHER.DATA] TO WR[0]      ;SET OTHER DATA PRINTOUT BIT
U 11E5, 3E80,15 ;15311          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL WORD TO PRINT ADDRESS
                ;15312                                           ; UNDER 'OTHER' IF ERROR
U 11E6, 3644,15 ;15313          MOV LS[#4] TO WR[0]              ;4 IN WR
U 11E7, BE82,15 ;15314          MOV WR[0] TO LS[ERROR.NUMBER]    ;ERROR 4
                ;15315  LOOP.T21.4:
U 11E8, 2F82,95 ;15316          CLR WR[1]                        ;EXPECTED DATA
```

```
U 11E9, 1813,F5 ;15317          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;15318                                  ;SET UP TO READ ZEROS
U 11EA, 3022,15 ;15319          MOV MEM.DATA TO WR[0]   ;GET RESULT FROM MEMORY
U 11EB, 0869,3C ;15320          JSR [CHECK.RESULT]      ;CHECK FOR ZEROS
U 11EC, 891E,84 ;15321          JMP [LOOP.T21.4]        ;LOOP ON ERROR IF ENABLED
                ;15322
U 11ED, B680,15 ;15323          MOV LS[CONTROL.STATUS] TO WR[0] ;GET CONTROL AND STATUS WORD
                ;15324          BIT LS[BIT8] WITH WR[0],        ;SEE IF AN ERROR OCCURRED (BIT 8 WILL
                ;15325                                          ; HAVE SET IN THE CHECK.RESULT ROUTINE)
U 11EE, D950,35 ;15326           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 11EF, C550,15 ;15327          BIC LS[BIT8] TO WR[0]           ;CLEAR BIT 8 IF SET
U 11F0, 3E80,15 ;15328          MOV WR[0] TO LS[CONTROL.STATUS] ;RESTORE CONTROL WORD WITH BIT 8 CLEAR
U 11F1, 091F,B1 ;15329          JMP.IF[BIT.SET] TO [T21.WRITE.4.COMP] ;IF ERROR, DON'T CHECK FOR ERROR
                ;15330                                          ; SUM
                ;15331
U 11F2, 5B00,1D ;15332          SKIP.IF[MEM.REF.OK]             ;ELSE CHECK FOR ERROR SUM
U 11F3, 0923,9C ;15333          JSR [T21.5.ERRORSUM]            ;GO SEE IF CRD OR OTHER ERROR
U 11F4, 091F,B4 ;15334          JMP [T21.WRITE.4.COMP]          ;HERE IF NO LOOP ON ERROR OR NO ERROR
                ;15335                                          ; SUM
U 11F5, 091E,34 ;15336          JMP [LOOP.T21.5]                ;ERROR LOOP IF ENABLED
                ;15337  LOOP.T21.5A:
U 11F6, E58A,15 ;15338          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
U 11F7, B670,15 ;15339          MOV LS[OTHER.DATA] TO WR[0]     ;SET OTHER DATA PRINTOUT BIT
U 11F8, 3E80,15 ;15340          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL WORD TO PRINT ADDRESS
                ;15341                                          ; UNDER 'OTHER' IF ERROR
U 11F9, 3644,15 ;15342          MOV LS[#4] TO WR[0]             ;4 IN WR
U 11FA, BE82,15 ;15343          MOV WR[0] TO LS[ERROR.NUMBER]   ;ERROR 4
                ;15344  T21.WRITE.4.COMP:
U 11FB, B67E,95 ;15345          MOV LS[#80000000] TO WR[1]      ;EXPECTED DATA
U 11FC, 9912,75 ;15346          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;15347                                          ;SET UP TO WRITE COMPLEMENT CKBTS IN
                ;15348                                          ; MEMORY
U 11FD, B27E,15 ;15349          WRITE.MEM LS[#80000000]         ;WRITE 80000000(H) IN MEMORY
U 11FE, 1813,F5 ;15350          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;15351                                          ;SET UP TO READ NEW DATA
U 11FF, 3022,15 ;15352          MOV MEM.DATA TO WP[0]           ;GET RESULT
U 1200, 0869,3C ;15353          JSR [CHECK.RESULT]              ;CHECK DATA FOR 80000000
U 1201, 091F,B4 ;15354          JMP [T21.WRITE.4.COMP]          ;ERROR LOOP IF ENABLED
                ;15355
U 1202, B680,15 ;15356          MOV LS[CONTROL.STATUS] TO WR[0] ;GET CONTROL AND STATUS WORD
                ;15357          BIT LS[BIT8] WITH WR[0],        ;SEE IF AN ERROR OCCURRED (BIT 8 WILL
                ;15358                                          ; HAVE SET IN THE CHECK.RESULT ROUTINE)
U 1203, D950,35 ;15359           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1204, C550,15 ;15360          BIC LS[BIT8] TO WR[0]           ;CLEAR BIT 8 IF SET
U 1205, 3E80,15 ;15361          MOV WR[0] TO LS[CONTROL.STATUS] ;RESTORE CONTROL WORD WITH BIT 8 CLEAR
U 1206, 0920,B1 ;15362          JMP.IF[BIT.SET] TO [T21.4.INC]  ;IF ERROR, DON'T CHECK FOR ERROR SUM. GO
                ;15363                                          ; DO NEXT ADDRESS
                ;15364
U 1207, 5B00,1D ;15365          SKIP.IF[MEM.REF.OK]             ;ELSE CHECK FOR ERROR SUM
U 1208, 0923,9C ;15366          JSR [T21.5.ERRORSUM]            ;GO SEE IF CRD OR OTHER ERROR
U 1209, 5B00,1E ;15367          SKIP                            ;HERE IF NO LOOP ON ERROR OR NO ERROR
                ;15368                                          ; SUM
U 120A, 891F,64 ;15369          JMP [LOOP.T21.5A]               ;ERROR LOOP IF ENABLED ON CSR 1 ERROR
                ;15370
                ;15371  T21.4.INC:
```

```
U 120B, 6C13,95 ;15372          ADD WR[3] TO LS[T9]              ;INCREMENT ADDRESS BY 4
U 120C, 6C89,95 ;15373          ADD WR[3] TO LS[ADDRESS.DATA]    ;ALSO INCREMENT ADDRESS TO PRINT
                ;15374          CMP WR[2] WITH LS[T9],           ;SEE IF DONE ALL ADDRESSES IN THIS BLOCK
U 120D, CF13,35 ;15375           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 120E, 891E,81 ;15376          JMP.IF[NEQ] TO [LOOP.T21.4]      ;REPEAT TEST AT NEW ADDRESS IF NOT DONE
                ;15377                                           ; ELSE PRINT SINGLE BIT ERRORS IN THIS
                ;15378                                           ; BLOCK AND START NEW 256KB BLOCK
U 120F, 3614,15 ;15379          MOV LS[T10] TO WR[0]             ;GET SINGLE BIT ERROR COUNT
U 1210, 3E0E,15 ;15380          MOV WR[0] TO LS[T7.SUB]          ;PUT NUMBER OF ERRORS IN LS 7 FOR PRINT-
                ;15381                                           ; OUT. BITS 24-31 CONTAIN BLOCK NUMBER
                ;15382                                           ; LS T15 CONTAINS ARRAY NUMBER (64K RAM)
U 1211, 3674,15 ;15383          MOV LS[PRINT.CRD] TO WR[0]       ;SET BIT 26 IN WR
U 1212, 3E80,15 ;15384          MOV WR[0] TO LS[CONTROL.STATUS]  ;TELL CONSOLE TO PRINT NUMBER OF CRD'S
                ;15385                                           ; ON CURRENT BLOCK
U 1213, 10E0,15 ;15386          MISC [SET.CP.ATTN]               ;ISSUE CPU ATTN TO CONSOLE
                ;15387  WAIT.T21.1:
U 1214, 8921,44 ;15388          JMP [WAIT.T21.1]                 ;LOOP HERE WAITING FOR CONSOLE TO
                ;15389                                           ; RESPOND
                ;15390
U 1215, 3624,15 ;15391          MOV LS[MM.LASTADDR+1] TO WR[0]   ;SEE IF DONE ALL AVAILABLE MEMORY
                ;15392          CMP WR[0] WITH LS[T9],           ;CURRENT ADDRESS=LAST ADDRESS + 1?
U 1216, 4F12,35 ;15393           DT(LONG)&SET.ALU.CC             ; SET CONDITION CODES
U 1217, 8927,99 ;15394          JMP.IF[EQL] TO [END.T21]         ;GO TO END IF DONE
                ;15395
U 1218, 3614,15 ;15396          MOV LS[T10] TO WR[0]             ;ELSE GET SINGLE BIT ERROR COUNT FROM
                ;15397                                           ; PREVIOUS BLOCK
U 1219, 5F2A,95 ;15398          MCOM LS[#FF000000] TO WR[1]      ;PUT FFFFFF(H) IN WR
U 121A, AF42,15 ;15399          BIC WR[1] TO WR[0]               ;CLEAR ERROR COUNT, BUT LEAVE BLOCK
                ;15400                                           ; NUMBER UNMODIFIED
U 121B, 4070,15 ;15401          ADD LS[BIT24] TO WR[0]           ;INCREMENT BLOCK NUMBER IN SINGLE BIT
                ;15402                                           ; ERROR COUNT
U 121C, 3670,95 ;15403          MOV LS[BIT24] TO WR[1]           ;SET BIT 24 IN WR
U 121D, C772,95 ;15404          BIS LS[BIT25] TO WR[1]           ;AND SET BIT 25 IN WR
                ;15405          BIT WR[1] WITH WR[0],            ;SEE IF EVEN BOUNDARY OF 4 (BOTH BITS
U 121E, A602,35 ;15406           DT(LONG)&SET.ALU.CC             ; CLEAR) AND SET CONDITION CODES
U 121F, 8922,21 ;15407          JMP.IF[BIT.SET] TO [STORE.T21]   ;JUMP IF EITHER OR BOTH BITS SET
U 1220, FF1E,15 ;15408          INC LS[T15]                      ;ELSE INCREMENT ARRAY NUMBER  NOTE:
                ;15409                                           ; DO NOT CHANGE THIS LS LOCATION FROM
                ;15410                                           ; T15!!! MONITOR LOOKS AT THIS LS FOR
                ;15411                                           ; ARRAY NUMBER!
U 1221, 2F80,15 ;15412          CLR WR[0]                        ;START BACK AT BLOCK 0
                ;15413  STORE.T21:
U 1222, BE14,15 ;15414          MOV WR[0] TO LS[T10]             ;STORE IN LS
U 1223, C065,15 ;15415          ADD LS[#40000] TO WR[2]          ;LAST ADDRESS OF NEXT 256KB BLOCK
U 1224, 0919,D4 ;15416          JMP [LOOP.T21.2]                 ;START MARCH ON NEW BLOCK
                ;15417
                ;15418  ;
                ;15419  ; SUBROUTINES FOR THIS TEST
                ;15420  ;
                ;15421
                ;15422  T21.2.ERRORSUM:
U 1225, FF82,15 ;15423          INC LS[ERROR.NUMBER]             ;ERROR 2
U 1226, E580,15 ;15424          CLR LS[CONTROL.STATUS]           ;DISABLE 'OTHER' PRINTOUT
U 1227, 373E,15 ;15425          MOV LS[#3F003FFF] TO WR[0]       ;MASK TO CHECK ERROR BITS ONLY IN CSR
U 1228, 3E8A,15 ;15426          MOV WR[0] TO LS[ERROR.MASK]      ;SET UP ERROR MASK
```

```
U 1229, 367C,95 ;15427          MOV LS[CRD] TO WR[1]             ;EXPECTED DATA (CRD SET, OTHERS CLEAR)
U 122A, 9D45,75 ;15428          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;15429                                           ;SET UP TO READ CSR
U 122B, 3022,15 ;15430          MOV MEM.DATA TO WR[0]            ;GET CONTENTS OF CSR 1
U 122C, 0869,3C ;15431          JSR [CHECK.RESULT]               ;CHECK FOR CRD SET, OTHERS CLEAR
U 122D, DB00,16 ;15432          RETURN+1                         ;RETURN + 1 TO PERFORM ERROR LOOP
                ;15433
U 122E, E58A,15 ;15434          CLR LS[ERROR.MASK]               ;CHECK ALL BITS
U 122F, FC82,15 ;15435          DEC LS[ERROR.NUMBER]             ;ERROR 1
U 1230, B670,15 ;15436          MOV LS[OTHER.DATA] TO WR[0]      ;SET OTHER DATA PRINTOUT BIT
U 1231, 3E80,15 ;15437          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL WORD TO PRINT ADDRESS
                ;15438                                           ; UNDER 'OTHER' IF ERROR
U 1232, 0924,DC ;15439          JSR [CNT.SYNDROME]               ;GO TO DETERMINE IF SINGLE ERROR ON DATA
                ;15440                                           ; BITS
                ;15441
U 1233, 0923,84 ;15442          JMP [NO.CRD.RETURN]              ;RETURNS HERE IF NOT A SINGLE BIT ERROR
U 1234, 8923,89 ;15443          JMP.IF[EQL] TO [NO.CRD.RETURN]  ;SKIP INC OF DATA ERROR IF ONLY ONE
                ;15444                                           ; SYNDROME WAS SET (INDICATES CKBT ERROR)
U 1235, FF14,15 ;15445          INC LS[T10]                      ;ELSE INCREMENT TEMPORARY LS LOCATION
                ;15446                                           ; (SINGLE BIT DATA ERROR COUNT)
U 1236, 8925,EC ;15447          JSR [CHECK.SER]                  ;SEE IF APT OR SER AND PRINT ERROR
                ;15448                                           ; IF EITHER IS SET
U 1237, 0926,5C ;15449          JSR [REPORT.SINGLE.DATA]         ;RETURN HERE IF SINGLE ERRORS TO BE
                ;15450                                           ; REPORTED AS REGULAR ERRORS
                ;15451  NO.CRD.RETURN:
U 1238, 5B00,14 ;15452          RETURN                           ;DONE
                ;15453
                ;15454
                ;15455  T21.5.ERRORSUM:
U 1239, FF82,15 ;15456          INC LS[ERROR.NUMBER]             ;ERROR 5
U 123A, E580,15 ;15457          CLR LS[CONTROL.STATUS]           ;DISABLE 'OTHER' PRINTOUT
U 123B, 373E,15 ;15458          MOV LS[#3F003FFF] TO WR[0]       ;MASK TO CHECK ERROR BITS ONLY IN CSR
U 123C, 3E8A,15 ;15459          MOV WR[0] TO LS[ERROR.MASK]      ;SET UP ERROR MASK
U 123D, 367C,95 ;15460          MOV LS[CRD] TO WR[1]             ;EXPECTED DATA (CRD SET, OTHERS CLEAR)
U 123E, 9D45,75 ;15461          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;15462                                           ;SET UP TO READ CSR
U 123F, 3022,15 ;15463          MOV MEM.DATA TO WR[0]            ;GET CONTENTS OF CSR 1
U 1240, 0869,3C ;15464          JSR [CHECK.RESULT]               ;CHECK FOR CRD SET, OTHERS CLEAR
U 1241, DB00,16 ;15465          RETURN+1                         ;RETURN + 1 TO PERFORM ERROR LOOP
                ;15466
U 1242, E58A,15 ;15467          CLR LS[ERROR.MASK]               ;CHECK ALL BITS
U 1243, FC82,15 ;15468          DEC LS[ERROR.NUMBER]             ;ERROR 4
U 1244, B670,15 ;15469          MOV LS[OTHER.DATA] TO WR[0]      ;SET OTHER DATA PRINTOUT BIT
U 1245, 3E80,15 ;15470          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL WORD TO PRINT ADDRESS
                ;15471                                           ; UNDER 'OTHER' IF ERROR
U 1246, 0924,DC ;15472          JSR [CNT.SYNDROME]               ;GO TO DETERMINE IF SINGLE ERROR ON DATA
                ;15473                                           ; BITS
                ;15474
U 1247, 0924,C4 ;15475          JMP [NO.CRD.RETURN.5]            ;RETURNS HERE IF NOT A SINGLE BIT ERROR
U 1248, 0924,C1 ;15476          JMP.IF[NEQ] TO [NO.CRD.RETURN.5];SKIP INC OF CKBT ERROR IF MORE THAN ONE
                ;15477                                           ; SYNDROME WAS SET (INDICATES DATA ERROR)
U 1249, FF14,15 ;15478          INC LS[T10]                      ;ELSE INCREMENT TEMPORARY LS LOCATION
                ;15479                                           ; (SINGLE BIT CKBT ERROR COUNT)
U 124A, 8925,EC ;15480          JSR [CHECK.SER]                  ;SEE IF APT OR SER AND PRINT ERROR
                ;15481                                           ; IF EITHER IS SET
```

```
U 124B, 8926,EC ;15482          JSR [REPORT.SINGLE.CKBT]        ;RETURN HERE IF SINGLE ERRORS TO BE
                ;15483                                          ; REPORTED AS REGULAR ERRORS
                ;15484  NO.CRD.RETURN.5:
U 124C, 5B00,14 ;15485          RETURN                          ;DONE
                ;15486
                ;15487
                ;15488  CNT.SYNDROME:
U 124D, 3650,15 ;15489          MOV LS[ERROR] TO WR[0]          ;SET BIT 8 IN WR
                ;15490          BIT WR[0] WITH LS[CONTROL.STATUS], ;SEE IF ERROR BIT SET IN CONTROL WORD
U 124E, 5980,55 ;15491           DT(SIZE)&SET.ALU.CC            ; AND SET CONDITION CODES
U 124F, 0925,D1 ;15492          JMP.IF[BIT.SET] TO [NO.CRD]     ;JUMP IF CRD WAS NOT SET IN CSR 1
U 1250, 9D9D,75 ;15493          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                ;15494                                          ;ELSE SET UP TO READ SYNDROME BITS
U 1251, 3022,15 ;15495          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 0
U 1252, 6518,15 ;15496          CLR LS[T12]                     ;COUNT OF SYNDROMES SET
U 1253, E5F8,15 ;15497          CLR LS[OS]                      ;CLEAR INDEX
                ;15498  REPEAT.SYN.CNT:
                ;15499          BIT LS[SHIFT.OS(4-0)] WITH WR[0], ;SEE IF THIS SYNDROME BIT IS SET
U 1254, D9F6,35 ;15500           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1255, DB00,01 ;15501          SKIP.IF[BIT.SET]                ;SKIP IF THIS SYNDROME BIT IS SET
                ;15502                                          ; (SYNDROMES COME BACK COMPLEMENTED)
U 1256, FF18,15 ;15503          INC LS[T12]                     ;ELSE INCREMENT SYNDROME SET COUNT
U 1257, 7FF8,15 ;15504          INC LS[OS]                      ;CHECK NEXT SYNDROME BIT
U 1258, B64E,95 ;15505          MOV LS[BIT7] TO WR[1]           ;SET BIT 7 IN WR 1
                ;15506          BIT WR[1] WITH LS[SHIFT.OS(4-0)], ;SEE IF TESTED ALL 7 SYNDROME BITS
U 1259, 59F6,B5 ;15507           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 125A, 8925,49 ;15508          JMP.IF[BITS.CLR] TO [REPEAT.SYN.CNT] ;REPEAT SYNDROME COUNT IF NOT DONE
                ;15509          DEC LS[T12],                    ;SUBTRACT 1 FROM SYNDROME COUNT
U 125B, 7C18,35 ;15510           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 125C, DB00,16 ;15511          RETURN+1                        ;RETURN+1 TO SINGLE ERROR COUNT ROUTINE
                ;15512  NO.CRD:
U 125D, 5B00,14 ;15513          RETURN                          ;RETURN IF NOT CRD ERROR
                ;15514
                ;15515  CHECK.SER:
U 125E, 3690,15 ;15516          MOV LS[ERROR.CONTROL] TO WR[0]  ;GET ERROR CONTROL WORD
U 125F, B648,95 ;15517          MOV LS[SER] TO WR[1]            ;SET SER BIT IN WR
U 1260, 474A,95 ;15518          BIS LS[APT.PRESENT] TO WR[1]    ;SET APT PRESENT BIT IN WR
                ;15519          BIT WR[1] WITH WR[0],           ;SEE IF EITHER BIT SET IN ERROR CONTROL
U 1261, A602,35 ;15520           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1262, 5B00,09 ;15521          SKIP.IF[BITS.CLR]               ;SKIP TO RETURN+1 IF NO SER OR APT
U 1263, 5B00,14 ;15522          RETURN                          ;RETURN TO PRINT ERROR REPORT
U 1264, DB00,16 ;15523          RETURN+1                        ;RETURN WITH NO ERROR REPORT
                ;15524
                ;15525  REPORT.SINGLE.DATA:
U 1265, 3672,15 ;15526          MOV LS[ECC.DIS] TO WR[0]        ;SET ECC DISABLE IN WR
U 1266, 8A17,FC ;15527          JSR [WRITE.CSR1]                ;SET ECC DIS IN CSR 1
                ;15528  LOOP.SINGLE:
U 1267, 3616,95 ;15529          MOV LS[T11] TO WR[1]            ;EXPECTED DATA
U 1268, 1813,F5 ;15530          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;15531                                          ;SET UP TO READ FAILING LOCATION WITH
                ;15532                                          ; CORRECTION DISABLED
U 1269, 3022,15 ;15533          MOV MEM.DATA TO WR[0]           ;GET UNCORRECTED DATA FROM MEMORY
U 126A, 0869,3C ;15534          JSR [CHECK.RESULT]              ;PRINT ERROR REPORT
U 126B, 0926,74 ;15535          JMP [LOOP.SINGLE]               ;ERROR LOOP IF ENABLED
                ;15536
```

```
U 126C, 0A17,EC ;15537          JSR [CLEAR.CSR1]                ;CLEAR CSR 1 TO ENABLE CORRECTION AGAIN
U 126D, 5B00,14 ;15538          RETURN                          ;DONE
                ;15539
                ;15540  REPORT.SINGLE.CKBT:
U 126E, B712,15 ;15541          MOV LS[#FFFFFF80] TO WR[0]      ;SET BITS 7-31 IN WR
U 126F, 3E8A,15 ;15542          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK BITS 0 THROUGH 6 ONLY
                ;15543  LOOP.SINGLE.CKBT:
U 1270, 1813,F5 ;15544          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;15545                                          ;SET UP TO READ FAILING LOCATION
U 1271, 3022,15 ;15546          MOV MEM.DATA TO WR[0]           ;COMPLETE DATA CYCLE
U 1272, 369E,95 ;15547          MOV LS[ONES] TO WR[1]           ;FORCE ERROR (EXPECTED CAN NOT BE ALL
                ;15548                                          ; ONES, SYNDROMES COME BACK COMPLEMENTED)
U 1273, 9D9D,75 ;15549          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                ;15550                                          ;SET UP TO READ SYNDROME BITS
U 1274, 3022,15 ;15551          MOV MEM.DATA TO WR[0]           ;GET SYNDROMES
U 1275, 0869,3C ;15552          JSR [CHECK.RESULT]              ;REPORT RECEIVED SYNDROME
U 1276, 0927,04 ;15553          JMP [LOOP.SINGLE.CKBT]          ;ERROR LOOP IF ENABLED
U 1277, E58A,15 ;15554          CLR LS[ERROR.MASK]              ;RESTORE CHECK OF ALL BITS
U 1278, 5B00,14 ;15555          RETURN                          ;DONE
                ;15556
                ;15557  END.T21:
```

```
;15558  .PAGE   " Other Memory Controller Logic"
;15559
;15560  .TOC    "  TEST 22 NXM On Access To Non-Existant Unibus Device (M8391  MCT Module)"
;15561  ;++
;15562  ;
;15563  ; Test Description:
;15564  ;
;15565  ;       This test checks the logic associated with forcing an NXM when a UNIBUS
;15566  ;       timeout occurs.  The NXM bit will set on a read or write to a non-
;15567  ;       existant UNIBUS device, or a device that does not respond.  This test
;15568  ;       will first issue a write to address FFFFFE which is a non-existant
;15569  ;       UNIBUS address (there are no devices which respond to this address).
;15570  ;       The NXM bit is checked to assure that it is set.  Then a read to the
;15571  ;       same address is performed and the NXM bit is checked to assure that it
;15572  ;       is again set.
;15573  ;         The NXM is set on a read or write by the memory micro-code asserting
;15574  ;       ROT C0 and clocking the CSR.  ROT C0 is ORed with NXM at the CSR 1A
;15575  ;       PAL.  The micro-code will only assert ROT C0 if a UNIBUS timeout occurs
;15576  ;       twice before a SSYNC occurs.  The signal TIMEOUT L is produced each
;15577  ;       time REF IN PROG L toggles from a low to a high, and remains asserted
;15578  ;       for 1 memory cycle.  The TIMEOUT L signal goes to the branch logic for
;15579  ;       the memory u-code to branch on.  If timeout comes twice before SSYN,
;15580  ;       the NXM will set.
;15581  ;
;15582  ; Assumptions:
;15583  ;
;15584  ;       It is assumed that all previous tests have run successfully.
;15585  ;
;15586  ; Test Steps:
;15587  ;
;15588  ;       1)  Set up error number, and module number in LS (for error printouts)
;15589  ;           and clear previous error number in LS.
;15590  ;       2)  Set up error mask to check the NXM bit (bit 16).
;15591  ;       3)  Issue write to address FFFFFE and read CSR 1 for NXM set.
;15592  ;       4)  Issue read to address FFFFFE and read CSR 1 checking for NXM set.
;15593  ;
;15594  ; Errors:
;15595  ;
;15596  ;       Error 1 - NXM not set on write to non-existant UNIBUS device.
;15597  ;       Error 2 - NXM not set on read from non-existant UNIBUS device.
;15598  ;
;15599  ; Debug:
;15600  ;
;15601  ;       Two methods of debug will be explained.  If the MCT module is in a
;15602  ;       test station then the MCT can be single stepped making debug easier
;15603  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;15604  ;       method will be explained also.  When single stepping the memory con-
;15605  ;       troller will help in debug, this section will explicitly suggest it.
;15606  ;
;15607  ;       Error 1 - This error indicates a problem with the ROT C0 signal or the
;15608  ;       timeout logic which also controls the branch logic.  The signal TIMEOUT
;15609  ;       L can be checked while the memory is in an idle loop, since it comes
;15610  ;       from the refresh logic.  Check the signal at the input to the BEN 0
;15611  ;       MUX.  It should be pulsing low for 1 cycle once every 12 usec (approx).
;15612  ;       If TIMEOUT L is not pulsing, check it at the output of POWER UP AND
```

```
                ;15613  ;       INIT PAL.  If incorrect there, check the inputs for a pulse on REF IN
                ;15614  ;       PROG L.  If this is OK, the PAL is bad.
                ;15615  ;         If TIMEOUT L is OK, then check that ROT C0 H is going high while
                ;15616  ;       looping on error.  If not, the BEN 0 MUX may be bad so that the branch
                ;15617  ;       on TIMEOUT L is not occuring.
                ;15618  ;         If ROT C0 is OK, check that it gets through the inverter and OR gate
                ;15619  ;       to the CSR 1A PAL.
                ;15620  ;
                ;15621  ;       Error 2 - Same as error 1.
                ;15622  ;--
                ;15623
                ;15624  T.22:
                ;15625
U 1279, B65E,15 ;15626          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;15627                                          ; STATUS WORD
U 127A, 3E80,15 ;15628          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;15629                                          ; BIT 15 INDICATES START OF TEST TO
                ;15630                                          ; CONSOLE
U 127B, 10E0,15 ;15631          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;15632  WAIT.T22.0:
U 127C, 0927,C4 ;15633          JMP [WAIT.T22.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;15634                                          ; RESPOND
                ;15635
U 127D, 0A1A,9C ;15636          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                ;15637                                          ; CLEAR MCT CSR 1
                ;15638
U 127E, 5F60,15 ;15639          MCOM LS[NXM] TO WR[0]           ;CLEAR BIT 16 IN WR
U 127F, 3E8A,15 ;15640          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK NXM BIT (BIT 16 IN CSR 1) ONLY
U 1280, DF2A,15 ;15641          MCOM LS[#FF000000] TO WR[0]     ;PUT FFFFFF(H) IN WR 0
U 1281, 2100,15 ;15642          DEC WR[0]                       ;NOW HAVE FFFFFE(H)
U 1282, BE12,15 ;15643          MOV WR[0] TO LS[T9]              ;ADDRESS IN LS
                ;15644
                ;15645  LOOP.T22.1:
U 1283, B660,95 ;15646          MOV LS[NXM] TO WR[1]            ;EXPECTED DATA (NXM SET)
U 1284, 1912,35 ;15647          MEM.REQ[WRITE.P] ADRS[T9] DT[WORD]
                ;15648                                          ;SET UP TO WRITE TO NON-EXISTANT DEVICE
U 1285, 329E,15 ;15649          WRITE.MEM LS[ONES]              ;WRITE ONES AT NON-EXISTANT UNIBUS
                ;15650                                          ; ADDRESS (WILL TIMEOUT)
U 1286, 9D45,75 ;15651          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;15652                                          ;SET UP TO READ CSR 1
U 1287, 3022,15 ;15653          MOV MEM.DATA TO WR[0]           ;CONTENTS OF CSR 1 IN WR
U 1288, 0869,3C ;15654          JSR [CHECK.RESULT]              ;CHECK FOR NXM SET
U 1289, 0928,34 ;15655          JMP [LOOP.T22.1]                ;ERROR LOOP IF ENABLED
                ;15656
U 128A, FF82,15 ;15657          INC LS[ERROR.NUMBER]            ;ERROR 2
                ;15658  LOOP.T22.2:
U 128B, B660,95 ;15659          MOV LS[NXM] TO WR[1]            ;EXPECTED DATA (NXM SET)
U 128C, 9813,B5 ;15660          MEM.REQ[READ.P] ADRS[T9] DT[WORD]
                ;15661                                          ;SET UP TO READ NON-EXISTANT DEVICE
U 128D, 3022,15 ;15662          MOV MEM.DATA TO WR[0]           ;READ ONES AT NON-EXISTANT UNIBUS
                ;15663                                          ; ADDRESS (WILL TIMEOUT)
U 128E, 9D45,75 ;15664          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;15665                                          ;SET UP TO READ CSR 1
U 128F, 3022,15 ;15666          MOV MEM.DATA TO WR[0]           ;CONTENTS OF CSR 1 IN WR
U 1290, 0869,3C ;15667          JSR [CHECK.RESULT]              ;CHECK FOR NXM SET (NXM SETS ON READ
```

```
                ;15668                                  ; IF TIMEOUT)
U 1291, 8928,B4 ;15669          JMP [LOOP.T22.2]        ;ERROR LOOP IF ENABLED
                ;15670  END.T22:
```

```
;15671  .PAGE   "  TEST 23 Abort Instruction On CS Parity Error (M8391  MCT or M8390 CPU Module)"
;15672  ;++
;15673  ;
;15674  ; Test Description:
;15675  ;
;15676  ;       This test checks the logic that forces the MCT micro-code to address
;15677  ;       100(H) if a parity error occurs on the CPU u-code during the dispatch.
;15678  ;       A CPU ATTN is issued with bit 9 of the control and status word set
;15679  ;       which requests the monitor to create a parity error in WCS at the loc-
;15680  ;       ation specified in LS 7.  LS 7 was loaded previously with the address
;15681  ;       of a MEM.REQ used in this test.  The test first writes TB 0 with all
;15682  ;       ones in bits 25-29.  Then it issues a MEM.REQ with the parity error to
;15683  ;       write TB 0 with all zeros.  The parity error should cause the memory
;15684  ;       micro-code to abort the dispatch and go to the power up routine at ad-
;15685  ;       dress 100(H).  Then a check is made on TB 0 to make sure it was not
;15686  ;       written.  The monitor will ignore the parity error rather than report-
;15687  ;       ing it as an error.
;15688  ;
;15689  ; Assumptions:
;15690  ;
;15691  ;       It is assumed that all previous tests have run successfully.
;15692  ;
;15693  ; Test Steps:
;15694  ;
;15695  ;       1)  Set up error number, and module number in LS (for error printouts)
;15696  ;           and clear previous error number in LS.
;15697  ;       2)  Load error mask with data to check bits 25-29 only.
;15698  ;       3)  Load LS 7 with the address of the MEM.REQ in step 5 below, then
;15699  ;           issue CPU ATTN with bit 9 set to create a parity error in that
;15700  ;           u-word.
;15701  ;       4)  Write TB 0 with 1's in bits 22-1 and check.
;15702  ;       5)  Issue the MEM.REQ with the parity error with MF=WRITE.TB and DT=
;15703  ;           LONGWORD.
;15704  ;       6)  Issue a WRITE.MEM LS with LS of 0's.  This instruction will only
;15705  ;           write 1's in the TB if the parity error logic fails to force the MCT
;15706  ;           to go to the power up initialization.
;15707  ;       7)  Read TB 0 and check that bits 22-1 are still 0's.
;15708  ;       8)  Restore good parity to the micro-word in step 5 and reenable parity
;15709  ;           checking.
;15710  ;
;15711  ; Errors:
;15712  ;
;15713  ;       NOTE:  Expected and received data is TB 0 data bits 22 through 1.
;15714  ;
;15715  ;       Error 1 - Initial write to TB 0 failed.
;15716  ;       Error 2 - Parity error failed to force MCT to power up initialization.
;15717  ;
;15718  ; Debug:
;15719  ;
;15720  ;       Two methods of debug will be explained.  If the MCT module is in a
;15721  ;       test station then the MCT can be single stepped making debug easier
;15722  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;15723  ;       method will be explained also.  When single stepping the memory con-
;15724  ;       troller will help in debug, this section will explicitly suggest it.
;15725  ;
```

```
                ;15726  ;       Error 1 - This logic was tested in previous tests.  If this failure oc-
                ;15727  ;       curs, run previous tests again.
                ;15728  ;
                ;15729  ;       Error 2 - This error indicates a failure of the parity error logic on
                ;15730  ;       the MCT module or coming over from the CPU module.  Single step the CPU
                ;15731  ;       to the MEM.REQ instruction which contains the parity error.  The memory
                ;15732  ;       should return to the idle loop.  If not, check the signal CS PAR ERR H
                ;15733  ;       coming into the inverter.  If this signal is not high, trace it back
                ;15734  ;       to its source on the CPU module.
                ;15735  ;         If CS PAR ERR H is high at the inverter, check it at pin 1 of the
                ;15736  ;       AND-OR-INVERT gate for a low.  If this is OK, and MCT single step is
                ;15737  ;       available, enable MCT single step and step the CPU again to the MEM
                ;15738  ;       .REQ instruction with the parity error.  Step the MCT once so that it
                ;15739  ;       is at the START.ADDR.PROM instruction and check for a high on STOP MEM
                ;15740  ;       H.  If this is incorrect, check for a low at both CS PERR L and DISP EN
                ;15741  ;       L coming into the chip.  If these are low, the chip is faulty.
                ;15742  ;         If STOP MEM H is high, check the signal ADRS 8 L for a low.  If OK,
                ;15743  ;       check the enables on the BEN x MUXes for highs.
                ;15744  ;         If these are OK, check the enable on the mux which provides prom ad-
                ;15745  ;       dress bits 4 through 7.  The enable should be high.  This can be traced
                ;15746  ;       back if incorrect.
                ;15747  ;         Finally check the address outputs.  All should be in a high impedence
                ;15748  ;       state (unasserted) except for ADRS 8 L which should be low (asserted).
                ;15749  ;
                ;15750  ;--
                ;15751
                ;15752  T.23:
                ;15753
U 1292, B65E,15 ;15754          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;15755                                          ; STATUS WORD
U 1293, 3E80,15 ;15756          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;15757                                          ; BIT 15 INDICATES START OF TEST TO
                ;15758                                          ; CONSOLE
U 1294, 10E0,15 ;15759          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;15760  WAIT.T23.0:
U 1295, 8929,54 ;15761          JMP [WAIT.T23.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;15762                                          ; RESPOND
                ;15763
U 1296, 0A1A,9C ;15764          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                ;15765                                          ; CLEAR MCT CSR 1
U 1297, 4742,15 ;15766          BIS LS[CPU] TO WR[0]            ;ALSO SET BIT 1
U 1298, 3E8C,15 ;15767          MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                ;15768                                          ; MCT OR CPU BOARD
                ;15769
U 1299, B62A,15 ;15770          MOV LS[#FF000000] TO WR[0]      ;SET TOP BYTE FOR ERROR MASK
U 129A, C76E,15 ;15771          BIS LS[BIT23] TO WR[0]          ;ALSO SET BIT 23
U 129B, C740,15 ;15772          BIS LS[BIT0] TO WR[0]           ;AND BIT 0
U 129C, 3E8A,15 ;15773          MOV WR[0] TO LS[ERROR.MASK]     ;NOW BITS 22 TRHOUGH 22 WILL BE CHECKED
                ;15774                                          ; FOR ERRORS
U 129D, B658,15 ;15775          MOV LS[#1000] TO WR[0]          ;1000 IN WR
U 129E, C752,15 ;15776          BIS LS[#200] TO WR[0]           ;1200 IN WR
U 129F, 474E,15 ;15777          BIS LS[#80] TO WR[0]            ;1280 IN WR
U 12A0, C74C,15 ;15778          BIS LS[#40] TO WR[0]            ;12C0 IN WR
                ;15779
U 12A1, 3E0E,15 ;15780          MOV WR[0] TO LS[T7.SUB]         ;LOAD LS 7 WITH ADDRESS OF MEM.REQ TO
```

```
                 ;15781                                          ; LOAD WITH BAD PARITY
U 12A2, B652,15  ;15782          MOV LS[SET.PA.ERR] TO WR[0]     ;SET BIT 9 IN WR
U 12A3, 3E80,15  ;15783          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 9 IN CONTROL WORD.  BIT 9 SET
                 ;15784                                          ; TELLS MONITOR TO WRITE BAD PARITY IN
                 ;15785                                          ; WCS WORD SPECIFIED IN LS 7
U 12A4, 10E0,15  ;15786          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                 ;15787  WAIT.T23.1:
U 12A5, 892A,54  ;15788          JMP [WAIT.T23.1]                ;LOOP HERE WAITING FOR CONSOLE TO
                 ;15789                                          ; RESPOND
                 ;15790
                 ;15791  LOOP.T23.1:
U 12A6, 369E,95  ;15792          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 12A7, 989C,F5  ;15793          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                 ;15794                                          ;SET UP TO WRITE TB ADDRESS 0
U 12A8, 329E,15  ;15795          WRITE.MEM LS[ONES]              ;WRITE ALL ONES IN TB 0
U 12A9, 9C9D,F5  ;15796          MEM.REQ[READ.TB] ADRS[#0] DT[LONG]
                 ;15797                                          ;SET UP TO READ TB 0
U 12AA, 3022,15  ;15798          MOV MEM.DATA TO WR[0]           ;READ CONTENTS OF TB 0
U 12AB, 0869,3C  ;15799          JSR [CHECK.RESULT]              ;CHECK THAT TB WAS WRITTEN CORRECTLY
U 12AC, 892A,64  ;15800          JMP [LOOP.T23.1]                ;ERROR LOOP IF ENABLED
                 ;15801
U 12AD, FF82,15  ;15802          INC LS[ERROR.NUMBER]            ;ERROR 2
                 ;15803  LOOP.T23.2:
U 12AE, 369E,95  ;15804          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA (NO CHANGE IN TB)
U 12AF, 892C,04  ;15805          JMP [T23.PAR.ERR]               ;GO TO MEM.REQUEST WITH PARITY ERROR
                 ;15806  12C0:
                 ;15807  T23.PAR.ERR:
U 12C0, 989C,F5  ;15808          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                 ;15809                                          ;SET UP TO ATTEMPT WRITE TO TB
U 12C1, B29C,15  ;15810          WRITE.MEM LS[ZERO]              ;TRY TO WRITE 0'S IN TB
U 12C2, 9C9D,F5  ;15811          MEM.REQ[READ.TB] ADRS[#0] DT[LONG]
                 ;15812                                          ;SET UP TO READ TB
U 12C3, 3022,15  ;15813          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF TB
U 12C4, 0869,3C  ;15814          JSR [CHECK.RESULT]              ;CHECK THAT TB IS UNCHANGED
U 12C5, 092A,E4  ;15815          JMP [LOOP.T23.2]                ;ERROR LOOP IF ENABLED
                 ;15816
U 12C6, B658,15  ;15817          MOV LS[#1000] TO WR[0]          ;1000 IN WR
U 12C7, C752,15  ;15818          BIS LS[#200] TO WR[0]           ;1200 IN WR
U 12C8, 474E,15  ;15819          BIS LS[#80] TO WR[0]            ;1280 IN WR
U 12C9, C74C,15  ;15820          BIS LS[#40] TO WR[0]            ;12C0 IN WR
U 12CA, 3E0E,15  ;15821          MOV WR[0] TO LS[T7.SUB]         ;LOAD LS 7 WITH ADDRESS OF MEM.REQ TO
                 ;15822                                          ; RELOAD WITH GOOD PARITY
U 12CB, B676,15  ;15823          MOV LS[CLR.PA.ERR] TO WR[0]     ;SET BIT 27 IN WR
U 12CC, 3E80,15  ;15824          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 27 IN CONTROL WORD.  BIT 27 SET
                 ;15825                                          ; TELLS MONITOR TO WRITE GOOD PARITY IN
                 ;15826                                          ; WCS WORD SPECIFIED IN LS 7
U 12CD, 10E0,15  ;15827          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                 ;15828  WAIT.T23.2:
U 12CE, 092C,E4  ;15829          JMP [WAIT.T23.2]                ;LOOP HERE WAITING FOR CONSOLE TO
                 ;15830                                          ; RESPOND
                 ;15831  END.T23:
```

```
;15832  .PAGE   "  TEST 24 March Test of CSR 1 Bits and ERROR SUM  (M8391  MCT Module)"
;15833  ;++
;15834  ;
;15835  ; Test Description:
;15836  ;
;15837  ;       This test checks that each CSR 1 error bit (except UBBSY, RDS and CRD)
;15838  ;       can be set independantly and that each bit will set the ERROR SUM bit
;15839  ;       by itself.  Also toe CLR ERR L logic that clears CSR 1 errors on any
;15840  ;       write to CSR 1 is tested.  The first part of the test also checks that
;15841  ;       ADDR PH will clear the error bits VALID, TB PAR ERROR, and TB MISS
;15842  ;       (these are CSR 1 PAL functions).  The test uses the same logic to set
;15843  ;       these bits as was used in previous tests, except error checking is done
;15844  ;       on all bits to assure that only one sets.  After each error bit is
;15845  ;       checked, a skip if memory referance ok is executed to check that ERROR
;15846  ;       SUM was set.
;15847  ;
;15848  ; Assumptions:
;15849  ;
;15850  ;       It is assumed that all previous tests have run successfully.
;15851  ;
;15852  ; Test Steps:
;15853  ;
;15854  ;       1)  Set up error number and module number in LS (for error printouts)
;15855  ;           and clear previous error number in LS.
;15856  ;       2)  Set up error mask to check bits 14 through 23.  Clear MODE bits in
;15857  ;           PSL to indicate KERNAL MODE.  Set MME and TB PAR DIAG (bits 27 and
;15858  ;           29) in CSR 1.
;15859  ;       3)  Write the TB at address 0 with bits 25 and 28 set.  This sets the
;15860  ;           BYTE OFFSET bit and the PROT to 2.
;15861  ;       4)  Execute TEST.V.RCHK with an address of F00000.  Execute MOV MEM.DAT
;15862  ;           to WR to complete the memory referance.
;15863  ;       5)  Read CSR 1 and check for VALID, TB MISS, and TB PAR ERR set, others
;15864  ;           clear.
;15865  ;       6)  Execute READ.P to address F00000.  Issue MOV MEM.DATA TO WR to com-
;15866  ;           plete the memory referance.  Check for UB ADAP REG SEL set, others
;15867  ;           clear.
;15868  ;       7)  Execute SKIP.IF(MEM.REF.OK) and check for no skip.  Write CSR 1 and
;15869  ;           check that all error bits are clear.
;15870  ;       8)  Execute READ.P to address 54000.  Issue MOV MEM.DATA TO WR to com-
;15871  ;           plete the memory referance.  Check for NXM set, others clear.  Then
;15872  ;           repeat step 7.
;15873  ;       9)  Execute READ.P to address FFFFFF.  Issue MOV MEM.DATA TO WR to com-
;15874  ;           plete the memory referance.  Check for ILL UB OPER set, others
;15875  ;           clear.  Then repeat step 7.
;15876  ;       10) Write TB 0 with data of 1 in BYTE OFFSET, VALID, MOD, and PROT of 2
;15877  ;           (bits 25, 26, 28, and 31 set).  Other bits are zero.
;15878  ;       11) Execute TEST.V.RCHK to address 0.  Issue MOV MEM.DATA TO WR to com-
;15879  ;           plete the memory referance.  Check for TB PAR ERR set, others
;15880  ;           clear.  Then repeat step 7.
;15881  ;       12) Clear the TB PAR DIAG bit in CSR 1 (but leave MME set).
;15882  ;       13) Execute TEST.V.WCHK to address 1FF. Issue MOV MEM.DATA TO WR to com-
;15883  ;           plete the memory referance.  Check for WR ACROSS PG ERR set, others
;15884  ;           clear.  Then repeat step 7.
;15885  ;       14) Write TB 0 with data of 1 in BYTE OFFSET, VALID, and PROT of 2
;15886  ;           (bits 25, 28, and 31 set).  Other bits are zero.
```

```
;15887  ;       15) Execute TEST.V.WCHK to address 0.  Issue MOV MEM.DATA TO WR to com-
;15888  ;           plete the memory referance.  Check for MODIFY REFUSED set, others
;15889  ;           clear.  Then repeat step 7.
;15890  ;       16) Write TB 0 with data of 1 in BYTE OFFSET and VALID (bits 25 and 31
;15891  ;           set).  Other bits are zero.
;15892  ;       17) Execute TEST.V.WCHK to address 0.  Issue MOV MEM.DATA TO WR to com-
;15893  ;           plete the memory referance.  Check for ACCESS REFUSED set, others
;15894  ;           clear.  Then repeat step 7.
;15895  ;       18) Write TB 0 with data of 1 in BYTE OFFSET and a PROT of 2 (bits 25
;15896  ;           and 28 set).  Other bits are zeros.
;15897  ;       19) Execute TEST.V.RCHK to address 0.  Issue MOV MEM.DATA TO WR to com-
;15898  ;           plete the memory referance.  Check for VALID set, others clear.
;15899  ;           Then repeat step 7.
;15900  ;       20) Write TB 0 with data of 1 in VALID and a PROT of 2 (bits 28 and 31
;15901  ;           set).  Other bits are 0.
;15902  ;       21) Execute TEST.V.RCHK to address 0.  Issue MOV MEM.DATA TO WR to com-
;15903  ;           plete the memory referance.  Check for TB MISS set, others clear.
;15904  ;           Then repeat step 7.
;15905  ;
;15906  ; Errors:
;15907  ;
;15908  ;       NOTE:  Expected and received data is CSR 1 error bits (bits 14-23).
;15909  ;
;15910  ;       Error 1 - Previously tested CSR bits failed, or shorted together.
;15911  ;       Error 2 - ADDR PH failed to clear virtual error bits in CSR, or
;15912  ;                 short between UB ADAP REG SEL bit and others.
;15913  ;       Error 3 - ERROR SUM not asserted on UB ADAP REG SEL.
;15914  ;       Error 4 - UB ADP REG SEL not cleared on write to CSR 1.
;15915  ;       Error 5 - Setting NXM set other bits.
;15916  ;       Error 6 - ERROR SUM not asserted on NXM.
;15917  ;       Error 7 - NXM not cleared on write to CSR 1.
;15918  ;       Error 8 - Setting ILL UB OPER set other bits.
;15919  ;       Error 9 - ERROR SUM not asserted on ILL UB OPER.
;15920  ;       Error A - UB ADP REG SEL not cleared on write to CSR 1.
;15921  ;       Error B - Setting TB PAR ERR set other bits.
;15922  ;       Error C - ERROR SUM not asserted on TB PAR ERR.
;15923  ;       Error D - TB PAR ERR not cleared on write to CSR 1.
;15924  ;       Error E - Setting WR ACROSS PG ERR set other bits.
;15925  ;       Error F - ERROR SUM not asserted on WR ACROSS PG ERR.
;15926  ;       Error 10- WR ACCROSS PG ERR not cleared on write to CSR 1.
;15927  ;       Error 11- Setting MODIFY REFUSED set other bits.
;15928  ;       Error 12- ERROR SUM not asserted on MODIFY REFUSED.
;15929  ;       Error 13- MODIFY REFUSED not cleared on write to CSR 1.
;15930  ;       Error 14- Setting ACCESS REFUSED set other bits.
;15931  ;       Error 15- ERROR SUM not asserted on ACCESS REFUSED.
;15932  ;       Error 16- ACCESS REFUSED not cleared on write to CSR 1.
;15933  ;       Error 17- Setting VALID set other bits.
;15934  ;       Error 18- ERROR SUM not asserted on VALID.
;15935  ;       Error 19- VALID not cleared on write to CSR 1.
;15936  ;       Error 1A- Setting TB MISS set other bits.
;15937  ;       Error 1B- ERROR SUM not asserted on TB MISS.
;15938  ;       Error 1C- TB MISS not cleared on write to CSR 1.
;15939  ;
;15940  ; Debug:
;15941  ;
```

```
                    ;15942  ;       Two methods of debug will be explained.  If the MCT module is in a
                    ;15943  ;       test station then the MCT can be single stepped making debug easier
                    ;15944  ;       in some cases.  Otherwise only the CPU can be single stepped and this
                    ;15945  ;       method will be explained also.  When single stepping the memory con-
                    ;15946  ;       troller will help in debug, this section will explicitly suggest it.
                    ;15947  ;
                    ;15948  ;       Error 1 - This error indicates a problem with previously tested logic
                    ;15949  ;       or a short between bits. Only bits 14, 15, and 21 should have been set.
                    ;15950  ;       If the previous tests all pass, suspect a short between bits or a prob-
                    ;15951  ;       lem with the CSR PAL that produces it.
                    ;15952  ;
                    ;15953  ;       Error 2 - If any bits besides bit 18 are set, suspect a problem with
                    ;15954  ;       either the CSR PAL that produces the bit, or the signal ADDR PH L at
                    ;15955  ;       the input to the CSR PAL.  If ADDR PH L fails to go low, bits 14, 15,
                    ;15956  ;       or 21 will not clear.  If bit 18 is not set, check for a short between
                    ;15957  ;       that bit and another.
                    ;15958  ;
                    ;15959  ;       Error 3,6,9,C,F,12,15,18,1B - Suspect the CSR PAL that produces the
                    ;15960  ;       error bit.
                    ;15961  ;
                    ;15962  ;       Error 4,7,A,D,10,13,16,19,1C - Suspect the CSR PAL that produces the
                    ;15963  ;       error bit or the input CLR ERR L.
                    ;15964  ;
                    ;15965  ;       Error 5,8,B,E,11,14,17,1A- Suspect a short between error bits or the
                    ;15966  ;       CSR PAL that produces the error bit.
                    ;15967  ;
                    ;15968  ;--
                    ;15969
                    ;15970  T.24:
                    ;15971
U 12CF, B65E,15     ;15972          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;15973                                          ; STATUS WORD
U 12D0, 3E80,15     ;15974          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;15975                                          ; BIT 15 INDICATES START OF TEST TO
                    ;15976                                          ; CONSOLE
U 12D1, 10E0,15     ;15977          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;15978  WAIT.T24.0:
U 12D2, 892D,24     ;15979          JMP [WAIT.T24.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;15980                                          ; RESPOND
                    ;15981
U 12D3, 0A1A,AC     ;15982          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    ;15983
U 12D4, 3628,15     ;15984          MOV LS[#FFFF] TO WR[0]          ;SET BITS 0-15 IN WR
U 12D5, C72A,15     ;15985          BIS LS[#FF000000] TO WR[0]      ;ALSO SET BITS 24-31 IN WR. NOW ONLY
                    ;15986                                          ; BITS 16-23 ARE CLEAR
U 12D6, C55C,15     ;15987          BIC LS[VALID.ERR] TO WR[0]      ;CLEAR BIT 14
U 12D7, 455E,15     ;15988          BIC LS[TB.PAR.ERR] TO WR[0]     ;AND BIT 15
U 12D8, 3E8A,15     ;15989          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK BITS 14-23 (CSR 1 ERROR BITS)
U 12D9, 2F80,15     ;15990          CLR WR[0]                       ;CLEAR WR
U 12DA, BFFE,15     ;15991          MOV WR[0] TO LS[PSL.HW]          ;CLEAR PSL TO MAKE CURRENT MODE=KERNEL
U 12DB, 0A17,DC     ;15992          JSR [WRITE.CSR1.MME]            ;ENABLE MEMORY MANAGEMENT
U 12DC, 3672,15     ;15993          MOV LS[BIT25] TO WR[0]          ;SET BIT 25 IN WR (USED TO SET BYTE
                    ;15994                                          ; OFFSET IN TB)
U 12DD, 4778,15     ;15995          BIS LS[BIT28] TO WR[0]          ;USED TO SET PROT TO 2 IN TB
U 12DE, BE14,15     ;15996          MOV WR[0] TO LS[T10]             ;PUT DATA IN LS
```

```
U 12DF, 6512,15 ;15997          CLR LS[T9]                          ;CLEAR ADDRESS IN LS
U 12E0, 9812,F5 ;15998          MEM.REQ[WRITE.TB] ADRS[T9] DT[LONG]
                ;15999                                              ;SET UP TO WRITE IN ADDRESS 0 OF TB
U 12E1, B214,15 ;16000          WRITE.MEM LS[T10]                   ;SET BYTE OFFSET AND PROT B IN TB
U 12E2, B676,15 ;16001          MOV LS[MME] TO WR[0]                ;SET BIT 27 IN WR
U 12E3, C77A,15 ;16002          BIS LS[TB.PAR.DIAG] TO WR[0]        ;ALSO SET BIT 29
U 12E4, 8A17,FC ;16003          JSR [WRITE.CSR1]                    ;ENABLE MEMORY MANAGEMENT AND TB PARITY
                ;16004                                              ; ERROR
U 12E5, B65D,15 ;16005          MOV LS[VALID.ERR] TO WR[2]          ;SET BIT 14 IN WR
U 12E6, C76B,15 ;16006          BIS LS[TB.MISS] TO WR[2]            ;ALSO SET BIT 21
U 12E7, 475F,15 ;16007          BIS LS[TB.PAR.ERR] TO WR[2]         ;AND BIT 15.  THIS IS EXPECTED DATA
U 12E8, B73A,15 ;16008          MOV LS[#F00000] TO WR[0]            ;SET UP UB ADAP REG ADDRESS IN WR
U 12E9, BE12,15 ;16009          MOV WR[0] TO LS[T9]                 ;SET UP LS TO ACCESS UB ADAP REG
                ;16010  LOOP.T24.1:
U 12EA, A004,95 ;16011          MOV WR[2] TO WR[1]                  ;EXPECTED DATA (VALID ERR, TB MISS, AND
                ;16012                                              ; TB PARITY ERR SET)
U 12EB, 9813,75 ;16013          MEM.REQ[TEST.V.RCHK] ADRS[T9] DT[LONG]
                ;16014                                              ;SET UP TO TEST AT TB ADDRESS 0
U 12EC, 3022,15 ;16015          MOV MEM.DATA TO WR[0]               ;ISSUE MOV TO COMPLETE U-CYCLE
U 12ED, 9D45,75 ;16016          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;16017                                              ;SET UP TO READ CSR1
U 12EE, 3022,15 ;16018          MOV MEM.DATA TO WR[0]               ;GET CSR 1 CONTENTS
U 12EF, 0869,3C ;16019          JSR [CHECK.RESULT]                  ;CHECK FOR VALID ERR,TB MISS,TB PAR ERR
U 12F0, 092E,A4 ;16020          JMP [LOOP.T24.1]                    ;ERROR LOOP IF ENABLED
                ;16021
U 12F1, 36C1,95 ;16022          MOV LS[#3(H)] TO WR[3]              ;ERROR 3 SAVED IN WR FOR RESTORE
                ;16023  LOOP.T24.3:
U 12F2, E580,15 ;16024          CLR LS[CONTROL.STATUS]              ;NORMAL ERROR PRINTOUT
U 12F3, BE83,95 ;16025          MOV WR[3] TO LS[ERROR.NUMBER]       ;ERROR 2
                ;16026  LOOP.T24.2:
U 12F4, 3664,95 ;16027          MOV LS[ADP.REG] TO WR[1]            ;EXPECTED DATA (UB ADAP REG SEL SET)
U 12F5, 1813,F5 ;16028          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;16029                                              ;SET UP TO READ AT PH ADDR F00000(H)
U 12F6, 3022,15 ;16030          MOV MEM.DATA TO WR[0]               ;ISSUE MOV TO COMPLETE U-CYCLE
U 12F7, 9D45,75 ;16031          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;16032                                              ;SET UP TO READ CSR1
U 12F8, 3022,15 ;16033          MOV MEM.DATA TO WR[0]               ;GET CSR 1 CONTENTS
U 12F9, 0869,3C ;16034          JSR [CHECK.RESULT]                  ;CHECK FOR UB ADAP REG SEL SET
U 12FA, 092F,44 ;16035          JMP [LOOP.T24.2]                    ;ERROR LOOP IF ENABLED
                ;16036
U 12FB, FF82,15 ;16037          INC LS[ERROR.NUMBER]                ;ERROR 3
U 12FC, 5B00,1D ;16038          SKIP.IF[MEM.REF.OK]                 ;CHECK FOR ERROR SUM (SHOULD NOT SKIP)
U 12FD, 0930,04 ;16039          JMP [TEST.T24.4]                    ;NO ERROR. GO TO NEXT PART OF TEST
U 12FE, 093A,BC ;16040          JSR [T24.NO.ERRSUM]                 ;REPORT NO ERROR SUM AS ERROR 4
U 12FF, 092F,24 ;16041          JMP [LOOP.T24.3]                    ;ERROR LOOP IF ENABLED
                ;16042
                ;16043  TEST.T24.4:
U 1300, 893B,1C ;16044          JSR [CHECK.CLRERR]                  ;CHECK THAT WRITE TO CSR CLEARS ERRORS
                ;16045                                              ; ERROR NUMBER 4
                ;16046
U 1301, 3645,95 ;16047          MOV LS[#4] TO WR[3]                 ;PUT A 4 IN WR 3
U 1302, 2047,95 ;16048          INC WR[3]                           ;ERROR 5 SAVED IN WR 3
U 1303, 3738,15 ;16049          MOV LS[#540000] TO WR[0]            ;NON EXISTANT MEMORY ADDRESS
U 1304, BE12,15 ;16050          MOV WR[0] TO LS[T9]                 ;STORE IN LS
                ;16051  LOOP.T24.6:
```

```
U 1305, E580.15 ;16052          CLR LS[CONTROL.STATUS]          ;NORMAL ERROR PRINTOUT
U 1306, BE83.95 ;16053          MOV WR[3] TO LS[ERROR.NUMBER]   ;ERROR 5
                ;16054  LOOP.T24.5:
U 1307, B660.95 ;16055          MOV LS[NXM] TO WR[1]            ;EXPECTED DATA (NXM SET)
U 1308, 1813.F5 ;16056          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;16057                                          ;SET UP TO READ AT PH ADDR 540000(H)
U 1309, 3022.15 ;16058          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE
U 130A, 9D45.75 ;16059          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;16060                                          ;SET UP TO READ CSR1
U 130B, 3022.15 ;16061          MOV MEM.DATA TO WR[0]           ;GET CSR 1 CONTENTS
U 130C, 0869.3C ;16062          JSR [CHECK.RESULT]              ;CHECK FOR NXM SET
U 130D, 8930.74 ;16063          JMP [LOOP.T24.5]                ;ERROR LOOP IF ENABLED
                ;16064
U 130E, FF82.15 ;16065          INC LS[ERROR.NUMBER]            ;ERROR 6
U 130F, 5B00.1D ;16066          SKIP.IF[MEM.REF.OK]             ;CHECK FOR ERROR SUM (SHOULD NOT SKIP)
U 1310, 8931.34 ;16067          JMP [TEST.T24.7]                ;NO ERROR. GO TO NEXT PART OF TEST
U 1311, 093A.BC ;16068          JSR [T24.NO.ERRSUM]             ;REPORT NO ERROR SUM AS ERROR 6
U 1312, 0930.54 ;16069          JMP [LOOP.T24.6]                ;ERROR LOOP IF ENABLED
                ;16070
                ;16071  TEST.T24.7:
U 1313, 893B.1C ;16072          JSR [CHECK.CLRERR]              ;CHECK THAT WRITE TO CSR CLEARS ERRORS
                ;16073                                          ; ERROR NUMBER 7
                ;16074  TEST.T24.8:
U 1314, C0C1.95 ;16075          ADD LS[#3(H)] TO WR[3]          ;ERROR 8 SAVED IN WR 3
U 1315, DF2A.15 ;16076          MCOM LS[#FF000000] TO WR[0]     ;NON EXISTANT UNIBUS DEVICE (ADDR FFFFFF)
U 1316, BE12.15 ;16077          MOV WR[0] TO LS[T9]             ;STORE IN LS
                ;16078  LOOP.T24.9:
U 1317, E580.15 ;16079          CLR LS[CONTROL.STATUS]          ;NORMAL ERROR PRINTOUT
U 1318, BE83.95 ;16080          MOV WR[3] TO LS[ERROR.NUMBER]   ;ERROR 8
                ;16081  LOOP.T24.8:
U 1319, 3668.95 ;16082          MOV LS[OP.ERR] TO WR[1]         ;EXPECTED DATA (ILL UB OPERATION ERR SET)
U 131A, 9813.B5 ;16083          MEM.REQ[READ.P] ADRS[T9] DT[WORD]
                ;16084                                          ;SET UP TO READ AT PH ADDR FFFFFF(H)
U 131B, 3022.15 ;16085          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE
U 131C, 9D45.75 ;16086          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;16087                                          ;SET UP TO READ CSR1
U 131D, 3022.15 ;16088          MOV MEM.DATA TO WR[0]           ;GET CSR 1 CONTENTS
U 131E, 0869.3C ;16089          JSR [CHECK.RESULT]              ;CHECK FOR OP ERR SET
U 131F, 8931.94 ;16090          JMP [LOOP.T24.8]                ;ERROR LOOP IF ENABLED
                ;16091
U 1320, FF82.15 ;16092          INC LS[ERROR.NUMBER]            ;ERROR 9
U 1321, 5B00.1D ;16093          SKIP.IF[MEM.REF.OK]             ;CHECK FOR ERROR SUM (SHOULD NOT SKIP)
U 1322, 8932.54 ;16094          JMP [TEST.T24.A]                ;NO ERROR. GO TO NEXT PART OF TEST
U 1323, 093A.BC ;16095          JSR [T24.NO.ERRSUM]             ;REPORT NO ERROR SUM AS ERROR 9
U 1324, 0931.74 ;16096          JMP [LOOP.T24.9]                ;ERROR LOOP IF ENABLED
                ;16097
                ;16098  TEST.T24.A:
U 1325, 893B.1C ;16099          JSR [CHECK.CLRERR]              ;CHECK THAT WRITE TO CSR CLEARS ERRORS
                ;16100                                          ; ERROR NUMBER A
                ;16101  TEST.T24.B:
U 1326, C0C1.95 ;16102          ADD LS[#3(H)] TO WR[3]          ;ERROR B SAVED IN WR 3
U 1327, 0A17.DC ;16103          JSR [WRITE.CSR1.MME]            ;ENABLE MEMORY MANAGEMENT
U 1328, 3672.15 ;16104          MOV LS[BIT25] TO WR[0]          ;BYTE OFFSET BIT
U 1329, 4774.15 ;16105          BIS LS[BIT26] TO WR[0]          ;MODIFY BIT
U 132A, 4778.15 ;16106          BIS LS[BIT28] TO WR[0]          ;PROT B BIT
```

```
U 132B, 477E,15 ;16107          BIS LS[BIT31] TO WR[0]          ;VALID BIT
U 132C, BE14,15 ;16108          MOV WR[0] TO LS[T10]            ;SET UP LS FOR TB WRITE
U 132D, 989C,F5 ;16109          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;16110                                          ;SET UP TO WRITE TB
U 132E, B214,15 ;16111          WRITE.MEM LS[T10]               ;SET BYTE OFFSET, MODIFY, PROT B, AND
                ;16112                                          ; VALID BITS IN TB ADDRESS 0
U 132F, B676,15 ;16113          MOV LS[MME] TO WR[0]            ;SET BIT 27 IN WR
U 1330, C77A,15 ;16114          BIS LS[TB.PAR.DIAG] TO WR[0]    ;ALSO SET BIT 29
U 1331, 8A17,FC ;16115          JSR [WRITE.CSR1]                ;ENABLE MEMORY MANAGEMENT AND TB PARITY
                ;16116                                          ; ERROR
                ;16117  LOOP.T24.C:
U 1332, E580,15 ;16118          CLR LS[CONTROL.STATUS]          ;NORMAL ERROR PRINTOUT
U 1333, BE83,95 ;16119          MOV WR[3] TO LS[ERROR.NUMBER]   ;ERROR B
                ;16120  LOOP.T24.B:
U 1334, 365E,95 ;16121          MOV LS[TB.PAR.ERR] TO WR[1]     ;EXPECTED DATA (TB PARITY ERROR SET)
U 1335, 989D,75 ;16122          MEM.REQ[TEST.V.RCHK] ADRS[#0] DT[LONG]
                ;16123                                          ;SET UP TO TEST AT TB ADDRESS 0
U 1336, 3022,15 ;16124          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE
U 1337, 9D45,75 ;16125          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;16126                                          ;SET UP TO READ CSR1
U 1338, 3022,15 ;16127          MOV MEM.DATA TO WR[0]           ;GET CSR 1 CONTENTS
U 1339, 0869,3C ;16128          JSR [CHECK.RESULT]              ;CHECK FOR TB PAR ERR SET
U 133A, 8933,44 ;16129          JMP [LOOP.T24.B]                ;ERROR LOOP IF ENABLED
                ;16130
U 133B, FF82,15 ;16131          INC LS[ERROR.NUMBER]            ;ERROR C
U 133C, 5B00,1D ;16132          SKIP.IF[MEM.REF.OK]             ;CHECK FOR ERROR SUM (SHOULD NOT SKIP)
U 133D, 8934,04 ;16133          JMP [TEST.T24.D]                ;NO ERROR. GO TO NEXT PART OF TEST
U 133E, 093A,BC ;16134          JSR [T24.NO.ERRSUM]             ;REPORT NO ERROR SUM AS ERROR B
U 133F, 8933,24 ;16135          JMP [LOOP.T24.C]                ;ERROR LOOP IF ENABLED
                ;16136
                ;16137  TEST.T24.D:
U 1340, 893B,1C ;16138          JSR [CHECK.CLRERR]              ;CHECK THAT WRITE TO CSR CLEARS ERRORS
                ;16139                                          ; ERROR NUMBER D
                ;16140  TEST.T24.E:
U 1341, C0C1,95 ;16141          ADD LS[#3(H)] TO WR[3]          ;ERROR E SAVED IN WR 3
U 1342, B626,15 ;16142          MOV LS[#FF] TO WR[0]            ;ADDRESS FF IN WR
U 1343, 4750,15 ;16143          BIS LS[#100] TO WR[0]           ;ADDRESS 1FF (PAGE BOUNDARY)
U 1344, BE12,15 ;16144          MOV WR[0] TO LS[T9]             ;STORE IN LS
U 1345, 0A17,DC ;16145          JSR [WRITE.CSR1.MME]            ;ENABLE MEMORY MANAGEMENT (CLEAR TB PAR
                ;16146                                          ; DIAG BIT)
                ;16147  LOOP.T24.F:
U 1346, E580,15 ;16148          CLR LS[CONTROL.STATUS]          ;NORMAL ERROR PRINTOUT
U 1347, BE83,95 ;16149          MOV WR[3] TO LS[ERROR.NUMBER]   ;ERROR E
                ;16150  LOOP.T24.E:
U 1348, B666,95 ;16151          MOV LS[WR.ACROSS.PG] TO WR[1]   ;EXPECTED DATA (WRITE ACROSS PAGE ERR SET)
U 1349, 1912,F5 ;16152          MEM.REQ[TEST.V.WCHK] ADRS[T9] DT[LONG]
                ;16153                                          ;SET UP TO TEST AT PAGE BOUNDARY
U 134A, 3022,15 ;16154          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE
U 134B, 9D45,75 ;16155          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;16156                                          ;SET UP TO READ CSR1
U 134C, 3022,15 ;16157          MOV MEM.DATA TO WR[0]           ;GET CSR 1 CONTENTS
U 134D, 0869,3C ;16158          JSR [CHECK.RESULT]              ;CHECK FOR WRITE ACROSS PAGE ERR SET
U 134E, 0934,84 ;16159          JMP [LOOP.T24.E]                ;ERROR LOOP IF ENABLED
                ;16160
U 134F, FF82,15 ;16161          INC LS[ERROR.NUMBER]            ;ERROR F
```

```
U 1350, 5B00,1D ;16162          SKIP.IF[MEM.REF.OK]             ;CHECK FOR ERROR SUM (SHOULD NOT SKIP)
U 1351, 8935,44 ;16163          JMP [TEST.T24.10]               ;NO ERROR. GO TO NEXT PART OF TEST
U 1352, 093A,BC ;16164          JSR [T24.NO.ERRSUM]             ;REPORT NO ERROR SUM AS ERROR F
U 1353, 8934,64 ;16165          JMP [LOOP.T24.F]                ;ERROR LOOP IF ENABLED
                ;16166
                ;16167  TEST.T24.10:
U 1354, 093B,CC ;16168          JSR [CHECK.CLRERR.NODIAG]       ;CHECK THAT WRITE TO CSR CLEARS ERRORS
                ;16169                                          ; ERROR NUMBER 10
                ;16170  TEST.T24.11:
U 1355, C0C1,95 ;16171          ADD LS[#3(H)] TO WR[3]          ;ERROR 11 SAVED IN WR 3
U 1356, 3672,15 ;16172          MOV LS[BIT25] TO WR[0]          ;BYTE OFFSET SET
U 1357, 4778,15 ;16173          BIS LS[BIT28] TO WR[0]          ;PROT B
U 1358, 477E,15 ;16174          BIS LS[BIT31] TO WR[0]          ;VALID SET
U 1359, BE14,15 ;16175          MOV WR[0] TO LS[T10]            ;STORE DATA IN LS
U 135A, 989C,F5 ;16176          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;16177                                          ;SET UP TO WRITE TB ADDRESS 0
U 135B, B214,15 ;16178          WRITE.MEM LS[T10]               ;SET BYTE OFFSET, PROT B, AND VALID IN
                ;16179                                          ; TB ADDRESS 0
                ;16180  LOOP.T24.12:
U 135C, E580,15 ;16181          CLR LS[CONTROL.STATUS]          ;NORMAL ERROR PRINTOUT
U 135D, BE83,95 ;16182          MOV WR[3] TO LS[ERROR.NUMBER]   ;ERROR 11
                ;16183  LOOP.T24.11:
U 135E, 366E,95 ;16184          MOV LS[MODIFY.REFUSED] TO WR[1] ;EXPECTED DATA (MODIFY REFUSED SET)
U 135F, 199C,F5 ;16185          MEM.REQ[TEST.V.WCHK] ADRS[#0] DT[LONG]
                ;16186                                          ;SET UP TO TEST AT TB ADDRESS 0
U 1360, 3022,15 ;16187          MOV MEM.DATA TO WR[0]           ;ISSUE MOV TO COMPLETE U-CYCLE
U 1361, 9D45,75 ;16188          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;16189                                          ;SET UP TO READ CSR1
U 1362, 3022,15 ;16190          MOV MEM.DATA TO WR[0]           ;GET CSR 1 CONTENTS
U 1363, 0869,3C ;16191          JSR [CHECK.RESULT]              ;CHECK FOR MODIFY REFUSED SET
U 1364, 8935,E4 ;16192          JMP [LOOP.T24.11]               ;ERROR LOOP IF ENABLED
                ;16193
U 1365, FF82,15 ;16194          INC LS[ERROR.NUMBER]            ;ERROR 12
U 1366, 5B00,1D ;16195          SKIP.IF[MEM.REF.OK]             ;CHECK FOR ERROR SUM (SHOULD NOT SKIP)
U 1367, 0936,A4 ;16196          JMP [TEST.T24.13]               ;NO ERROR. GO TO NEXT PART OF TEST
U 1368, 093A,BC ;16197          JSR [T24.NO.ERRSUM]             ;REPORT NO ERROR SUM AS ERROR 12
U 1369, 0935,C4 ;16198          JMP [LOOP.T24.12]               ;ERROR LOOP IF ENABLED
                ;16199
                ;16200  TEST.T24.13:
U 136A, 093B,CC ;16201          JSR [CHECK.CLRERR.NODIAG]       ;CHECK THAT WRITE TO CSR CLEARS ERRORS
                ;16202                                          ; ERROR NUMBER 13
                ;16203  TEST.T24.14:
U 136B, C0C1,95 ;16204          ADD LS[#3(H)] TO WR[3]          ;ERROR 14 SAVED IN WR 3
U 136C, 3672,15 ;16205          MOV LS[BIT25] TO WR[0]          ;BYTE OFFSET SET
U 136D, 477E,15 ;16206          BIS LS[BIT31] TO WR[0]          ;VALID BIT SET
U 136E, BE14,15 ;16207          MOV WR[0] TO LS[T10]            ;PUT DATA IN LS
U 136F, 989C,F5 ;16208          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;16209                                          ;SET UP TO WRITE TB ADDRESS 0
U 1370, B214,15 ;16210          WRITE.MEM LS[T10]               ;SET BYTE OFFSET AND VALID IN TB
                ;16211  LOOP.T24.15:
U 1371, E580,15 ;16212          CLR LS[CONTROL.STATUS]          ;NORMAL ERROR PRINTOUT
U 1372, BE83,95 ;16213          MOV WR[3] TO LS[ERROR.NUMBER]   ;ERROR 14
                ;16214  LOOP.T24.14:
U 1373, B66C,95 ;16215          MOV LS[ACCESS.REFUSED] TO WR[1] ;EXPECTED DATA (ACCESS REFUSED SET)
U 1374, 199C,F5 ;16216          MEM.REQ[TEST.V.WCHK] ADRS[#0] DT[LONG]
```

```
                    ;16217                                          ;TEST AT TB ADDRESS 0
U 1375, 3022,15     ;16218          MOV MEM.DATA TO WR[0]            ;ISSUE MOV TO COMPLETE U-CYCLE
U 1376, 9D45,75     ;16219          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;16220                                          ;SET UP TO READ CSR1
U 1377, 3022,15     ;16221          MOV MEM.DATA TO WR[0]            ;GET CSR 1 CONTENTS
U 1378, 0869,3C     ;16222          JSR [CHECK.RESULT]               ;CHECK FOR ACCESS REFUSED SET
U 1379, 8937,34     ;16223          JMP [LOOP.T24.14]                ;ERROR LOOP IF ENABLED
                    ;16224
U 137A, FF82,15     ;16225          INC LS[ERROR.NUMBER]             ;ERROR 15
U 137B, 5B00,1D     ;16226          SKIP.IF[MEM.REF.OK]              ;CHECK FOR ERROR SUM (SHOULD NOT SKIP)
U 137C, 8937,F4     ;16227          JMP [TEST.T24.16]                ;NO ERROR. GO TO NEXT PART OF TEST
U 137D, 093A,BC     ;16228          JSR [T24.NO.ERRSUM]              ;REPORT NO ERROR SUM AS ERROR 15
U 137E, 0937,14     ;16229          JMP [LOOP.T24.15]                ;ERROR LOOP IF ENABLED
                    ;16230
                    ;16231  TEST.T24.16:
U 137F, 093B,CC     ;16232          JSR [CHECK.CLRERR.NODIAG]        ;CHECK THAT WRITE TO CSR CLEARS ERRORS
                    ;16233                                          ; ERROR NUMBER 16
                    ;16234  TEST.T24.17:
U 1380, C0C1,95     ;16235          ADD LS[#3(H)] TO WR[3]           ;ERROR 17 SAVED IN WR 3
U 1381, 3672,15     ;16236          MOV LS[BIT25] TO WR[0]           ;BYTE OFFSET SET
U 1382, 4778,15     ;16237          BIS LS[BIT28] TO WR[0]           ;PROT B SET
U 1383, BE14,15     ;16238          MOV WR[0] TO LS[T10]             ;PUT DATA IN LS
U 1384, 989C,F5     ;16239          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                    ;16240                                          ;SET UP TO WRITE TB ADDRESS 0
U 1385, B214,15     ;16241          WRITE.MEM LS[T10]                ;SET BYTE OFFSET AND PROT B IN TB
                    ;16242  LOOP.T24.18:
U 1386, E580,15     ;16243          CLR LS[CONTROL.STATUS]           ;NORMAL ERROR PRINTOUT
U 1387, BE83,95     ;16244          MOV WR[3] TO LS[ERROR.NUMBER]    ;ERROR 17
                    ;16245  LOOP.T24.17:
U 1388, B65C,95     ;16246          MOV LS[VALID.ERR] TO WR[1]       ;EXPECTED DATA (VALID ERROR SET)
U 1389, 989D,75     ;16247          MEM.REQ[TEST.V.RCHK] ADRS[#0] DT[LONG]
                    ;16248                                          ;TEST AT TB ADDRESS 0
U 138A, 3022,15     ;16249          MOV MEM.DATA TO WR[0]            ;ISSUE MOV TO COMPLETE U-CYCLE
U 138B, 9D45,75     ;16250          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;16251                                          ;SET UP TO READ CSR1
U 138C, 3022,15     ;16252          MOV MEM.DATA TO WR[0]            ;GET CSR 1 CONTENTS
U 138D, 0869,3C     ;16253          JSR [CHECK.RESULT]               ;CHECK FOR VALID ERROR SET
U 138E, 0938,84     ;16254          JMP [LOOP.T24.17]                ;ERROR LOOP IF ENABLED
                    ;16255
U 138F, FF82,15     ;16256          INC LS[ERROR.NUMBER]             ;ERROR 18
U 1390, 5B00,1D     ;16257          SKIP.IF[MEM.REF.OK]              ;CHECK FOR ERROR SUM (SHOULD NOT SKIP)
U 1391, 8939,44     ;16258          JMP [TEST.T24.19]                ;NO ERROR. GO TO NEXT PART OF TEST
U 1392, 093A,BC     ;16259          JSR [T24.NO.ERRSUM]              ;REPORT NO ERROR SUM AS ERROR 18
U 1393, 8938,64     ;16260          JMP [LOOP.T24.18]                ;ERROR LOOP IF ENABLED
                    ;16261
                    ;16262  TEST.T24.19:
U 1394, 093B,CC     ;16263          JSR [CHECK.CLRERR.NODIAG]        ;CHECK THAT WRITE TO CSR CLEARS ERRORS
                    ;16264                                          ; ERROR NUMBER 19
                    ;16265  TEST.T24.1A:
U 1395, C043,95     ;16266          ADD LS[#2] TO WR[3]              ;ERROR 1A SAVED IN WR 3
U 1396, 3678,15     ;16267          MOV LS[BIT28] TO WR[0]           ;PROT B SET
U 1397, 477E,15     ;16268          BIS LS[BIT31] TO WR[0]           ;VALID SET
U 1398, BE14,15     ;16269          MOV WR[0] TO LS[T10]             ;STORE DATA IN LS
U 1399, 989C,F5     ;16270          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                    ;16271                                          ;SET UP TO WRITE TB ADDRESS 0
```

```
U 139A, B214,15 ;16272          WRITE.MEM LS[T10]                ;SET VALID AND PROT B IN TB
                ;16273  LOOP.T24.1B:
U 139B, E580,15 ;16274          CLR LS[CONTROL.STATUS]           ;NORMAL ERROR PRINTOUT
U 139C, BE83,95 ;16275          MOV WR[3] TO LS[ERROR.NUMBER]    ;ERROR 1B
                ;16276  LOOP.T24.1A:
U 139D, B66A,95 ;16277          MOV LS[TB.MISS] TO WR[1]         ;EXPECTED DATA (TB MISS SET)
U 139E, 989D,75 ;16278          MEM.REQ[TEST.V.RCHK] ADRS[#0] DT[LONG]
                ;16279                                           ;TEST AT TB ADDRESS 0
U 139F, 3022,15 ;16280          MOV MEM.DATA TO WR[0]            ;ISSUE MOV TO COMPLETE U-CYCLE
U 13A0, 9D45,75 ;16281          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;16282                                           ;SET UP TO READ CSR1
U 13A1, 3022,15 ;16283          MOV MEM.DATA TO WR[0]            ;GET CSR 1 CONTENTS
U 13A2, 0869,3C ;16284          JSR [CHECK.RESULT]               ;CHECK FOR TB MISS SET
U 13A3, 8939,D4 ;16285          JMP [LOOP.T24.1A]                ;ERROR LOOP IF ENABLED
                ;16286
U 13A4, FF82,15 ;16287          INC LS[ERROR.NUMBER]             ;ERROR 1B
U 13A5, 5B00,1D ;16288          SKIP.IF[MEM.REF.OK]              ;CHECK FOR ERROR SUM (SHOULD NOT SKIP)
U 13A6, 093A,94 ;16289          JMP [TEST.T24.1C]                ;JUMP TO NEXT TEST
U 13A7, 093A,BC ;16290          JSR [T24.NO.ERRSUM]              ;REPORT NO ERROR SUM AS ERROR 1B
U 13A8, 8939,B4 ;16291          JMP [LOOP.T24.1B]                ;ERROR LOOP IF ENABLED
                ;16292
                ;16293  TEST.T24.1C:
U 13A9, 093B,CC ;16294          JSR [CHECK.CLRERR.NODIAG]        ;CHECK THAT WRITE TO CSR CLEARS ERRORS
                ;16295                                           ; ERROR NUMBER 1C
U 13AA, 093C,64 ;16296          JMP [T24.END]                    ;DONE
                ;16297
                ;16298
                ;16299  T24.NO.ERRSUM:
U 13AB, B66E,15 ;16300          MOV LS[NA.EXP.REC] TO WR[0]      ;SET BIT 23 IN WR
U 13AC, 3E80,15 ;16301          MOV WR[0] TO LS[CONTROL.STATUS] ;PRINT N/A UNDER EXPECTED AND RECEIVED
                ;16302                                           ; DATA IN ERROR REPORT
U 13AD, 2F82,95 ;16303          CLR WR[1]                        ;WILL FORCE ERROR TO OCCUR
U 13AE, 0869,3C ;16304          JSR [CHECK.RESULT]               ;REPORT AN ERROR
U 13AF, 5B00,14 ;16305          RETURN                           ;ERROR LOOP RETURN
U 13B0, DB00,16 ;16306          RETURN+1                         ;NORMAL CONTINUE ON ERROR RETURN
                ;16307
                ;16308  CHECK.CLRERR:
U 13B1, E580,15 ;16309          CLR LS[CONTROL.STATUS]           ;NORMAL ERROR PRINTOUT
U 13B2, FF82,15 ;16310          INC LS[ERROR.NUMBER]             ;SET UP ERROR NUMBER
                ;16311  LOOP.CLRERR:
U 13B3, B676,15 ;16312          MOV LS[MME] TO WR[0]             ;SET BIT 27 IN WR
U 13B4, C77A,15 ;16313          BIS LS[TB.PAR.DIAG] TO WR[0]     ;ALSO SET BIT 29
U 13B5, 8A17,FC ;16314          JSR [WRITE.CSR1]                 ;ENABLE MEMORY MANAGEMENT AND TB PARITY
                ;16315                                           ; ERROR. WRITE TO CSR SHOULD CLEAR
                ;16316                                           ; ERROR BITS
U 13B6, 2F82,95 ;16317          CLR WR[1]                        ;EXPECTED DATA
U 13B7, 9D45,75 ;16318          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;16319                                           ;SET UP TO READ CSR1
U 13B8, 3022,15 ;16320          MOV MEM.DATA TO WR[0]            ;GET CSR 1 CONTENTS
U 13B9, 0869,3C ;16321          JSR [CHECK.RESULT]               ;CHECK FOR ALL ERROR BITS CLEARED
U 13BA, 893B,34 ;16322          JMP [LOOP.CLRERR]                ;ERROR LOOP IF ENABLED
U 13BB, 5B00,14 ;16323          RETURN                           ;BACK TO MAIN PROG
                ;16324
                ;16325  CHECK.CLRERR.NODIAG:
U 13BC, E580,15 ;16326          CLR LS[CONTROL.STATUS]           ;NORMAL ERROR PRINTOUT
```

```
U 13BD, FF82,15 ;16327          INC LS[ERROR.NUMBER]              ;SET UP ERROR NUMBER
                ;16328  LOOP.CLRERR2:
U 13BE, B676,15 ;16329          MOV LS[MME] TO WR[0]              ;SET BIT 27 IN WR
U 13BF, 8A17,FC ;16330          JSR [WRITE.CSR1]                  ;ENABLE MEMORY MANAGEMENT.  WRITE TO
                ;16331                                            ; CSR SHOULD CLEAR ERROR BITS
U 13C0, 2F82,95 ;16332          CLR WR[1]                         ;EXPECTED DATA
U 13C1, 9D45,75 ;16333          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;16334                                            ;SET UP TO READ CSR1
U 13C2, 3022,15 ;16335          MOV MEM.DATA TO WR[0]             ;GET CSR 1 CONTENTS
U 13C3, 0869,3C ;16336          JSR [CHECK.RESULT]                ;CHECK FOR ALL ERROR BITS CLEARED
U 13C4, 093B,E4 ;16337          JMP [LOOP.CLRERR2]                ;ERROR LOOP IF ENABLED
U 13C5, 5B00,14 ;16338          RETURN                            ;BACK TO MAIN PROG
                ;16339
                ;16340
                ;16341  T24.END:
```

```
;16342  .PAGE   "  TEST 25 Byte and Word Write To Memory (M8391  MCT Module)"
;16343  ;++
;16344  ;
;16345  ; Test Description:
;16346  ;
;16347  ;       This test checks the logic used to write byte or word data into main
;16348  ;       memory.  The MDR AND DIR CONTROL PAL is responsible for directing the
;16349  ;       new byte or word into the correct memory positions while leaving the
;16350  ;       surrounding data unchanged.  This test checks the MDR AND DIR CONTROL
;16351  ;       PAL extensively.
;16352  ;         The test will write bytes of 0 at each byte position in a longword of
;16353  ;       ones and check that only one byte is cleared.  It will then write words
;16354  ;       of 0 at each byte position within a longword of ones and check that 2
;16355  ;       bytes are cleared.  The write at byte position 3 will also check the
;16356  ;       following longword to assure that only the first byte is cleared.  Then
;16357  ;       a longword of 0's is written at position 0 (an aligned longword) with a
;16358  ;       software delay between the MEM.REQ and the WRITE.MEM instructions. This
;16359  ;       forces a different path in the micro-code than the array tests run pre-
;16360  ;       viously and checks another MDR AND DIR CONTROL PAL circuit used only
;16361  ;       for aligned longword writes that do not receive CPU DATA REQUEST within
;16362  ;       about 3 CPU cycles after the MEM.REQ (depending on if and when a re-
;16363  ;       fresh is required).  Next unaligned longword writes to positions 1, 2,
;16364  ;       and 3 are executed and 2 longwords are checked.
;16365  ;         Finally, a byte of 0's is written at byte position 0 in a longword of
;16366  ;       ones (as above) with a software delay between the MEM.REQ and the WRITE
;16367  ;       .MEM instructions.  This checks another path through the micro-code.
;16368  ;
;16369  ; Assumptions:
;16370  ;
;16371  ;       It is assumed that all previous tests have run successfully.
;16372  ;
;16373  ; Test Steps:
;16374  ;
;16375  ;       1)  Set up error mask, error number, and module number in LS (for
;16376  ;           error printouts) and clear previous error number in LS.
;16377  ;       2)  Load WR 2 with the memory size (LS location MM.LASTADDR+1) and
;16378  ;           check that size is not 0.
;16379  ;       3)  Write all 1's in memory location 0.
;16380  ;       4)  Write a byte of 0's at location 0.  Check result for byte 0 clear.
;16381  ;       5)  Repeat steps 3 and 4 at byte position 1, 2, and 3.
;16382  ;       6)  Repeat steps 3 through 5 with word data.  For the word write at
;16383  ;           byte position 3, prewrite locations 0 and 4 with 1's and check both
;16384  ;           longwords after the word write.
;16385  ;       7)  Repeat step 3, then issue a MEM.REQ with MF=WRITE.P and DT=longword
;16386  ;           using address 0 as the destination.
;16387  ;       8)  Delay 6 CPU cycles and issue a WRITE MEM.DATA LS instruction with
;16388  ;           data of all 0's.  Check location 0 for all 0's.
;16389  ;       9)  Write all 1's in memory locations 0 and 4 (two consecutive long-
;16390  ;           words).
;16391  ;       10) Write a longword of 0's at byte position 1.  Read back longword at
;16392  ;           locations 0 and 4 and check for correct data.
;16393  ;       11) Repeat steps 9 and 10 writing at byte positions 2 and 3.
;16394  ;       12) Write all 1's in memory location 0.
;16395  ;       13) Write a byte of 0's at location 0 with a software delay (NOPs) be-
;16396  ;           tween the MEM.REQ and the WRITE.MEM instructions.  Check result for
```

```
;16397  ;           byte 0 clear.                                                    U
;16398  ;                                                                           U
;16399  ; Errors:                                                                   U
;16400  ;
;16401  ;       NOTE:  OTHER data is array address being checked.
;16402  ;
;16403  ;       Error 1 - Byte write of 0's failed to write one and only one byte.
;16404  ;       Error 2 - Word write of 0's failed to write one and only one word
;16405  ;                 within longword location 0 (or byte 3 only in 2 cycle write).
;16406  ;       Error 3 - Two cycle word write failed to write 2nd byte of write data
;16407  ;                 at byte position 0 of location 4.
;16408  ;       Error 4 - Aligned longword write failed with delay between MEM.REQ and
;16409  ;                 WRITE.MEM LS.
;16410  ;       Error 5 - Unaligned longword write failed at location 0.
;16411  ;       Error 6 - Unaligned longword write failed at location 4.
;16412  ;       Error 7 - Byte write failed to write one and only one byte with delay
;16413  ;                 between MEM.REQ and WRITE.MEM LS.
;16414  ;
;16415  ; Debug:
;16416  ;
;16417  ;       Two methods of debug will be explained.  If the MCT module is in a
;16418  ;       test station then the MCT can be single stepped making debug easier
;16419  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;16420  ;       method will be explained also.  When single stepping the memory con-
;16421  ;       troller will help in debug, this section will explicitly suggest it.
;16422  ;
;16423  ;       Error 1 - This error indicates a failure in the MDR AND DIR CONTROL
;16424  ;       PAL or its inputs.  The expected data in the error report tells which
;16425  ;       byte the test was trying to write (the byte of 0's).  Single step the
;16426  ;       CPU to the MEM.REQ instruction with MF=WRITE.P and DT=BYTE.  Check the
;16427  ;       outputs of the MDR AND DIR CONTROL PAL for a low on the output which
;16428  ;       corresponds to the byte the test was trying to write.  Check the other
;16429  ;       3 outputs for highs.  If these are all correct, the micro-code may be
;16430  ;       faulty, or there may be a short between the signal LAT OUTPUT BYT L
;16431  ;       and one of the enables at the 'AND' gate that outputs LAT OUTPUT BYT x
;16432  ;       L.
;16433  ;         If the outputs out of the MDR AND DIR CONTROL PAL are bad, check the
;16434  ;       inputs for a high on ROT C0 H and CPH/UBL H.  Check for a low on 2ND
;16435  ;       MEM CYC H.  Check for a high on A1 L if we are writing to byte 0 or 1,
;16436  ;       check for a low on A1 L if we are writing to byte 2 or 3.  If these in-
;16437  ;       puts are all OK, then the PAL is probably bad.
;16438  ;         One other possibility is that the branch on ALIGN LW L is not working
;16439  ;       correctly.  This would result in all bytes being cleared instead of one
;16440  ;       byte.  It can be checked by single stepping to the same MEM.REQ as be-
;16441  ;       fore and scoping the input to the BEN 2 MUX.  It should be high.
;16442  ;
;16443  ;       Error 2 - If this is the first error, suspect the MDR AND DIR CONTROL
;16444  ;       PAL.
;16445  ;
;16446  ;       Error 3 - If this is the first error, the most likely suspect is the
;16447  ;       MDR AND DIR CONTROL PAL.  One input to this PAL that is different than
;16448  ;       the previous errors is that 2ND MEM CYC H must go high for the 2nd mem-
;16449  ;       ory cycle to work correctly.  While looping on error, check that 2ND
;16450  ;       MEM CYC H will go high.  If this is OK, suspect the PAL.
;16451  ;
```

```
                    ;16452  ;       Error 4 through 6 - Suspect the MDR AND DIR CONTROL PAL if this is the
                    ;16453  ;       first error.
                    ;16454  ;
                    ;16455  ;       Error 7 - Suspect the micro-code proms as this test takes a different
                    ;16456  ;       path through the micro-code.
                    ;16457  ;
                    ;16458  ;--
                    ;16459
                    ;16460  T.25:
                    ;16461
U 13C6, B65E,15     ;16462          MOV LS[BEGIN.TEST] TO WR[0]      ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;16463                                           ; STATUS WORD
U 13C7, 3E80,15     ;16464          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;16465                                           ; BIT 15 INDICATES START OF TEST TO
                    ;16466                                           ; CONSOLE
U 13C8, 10E0,15     ;16467          MISC [SET.CP.ATTN]               ;ISSUE CPU ATTN TO CONSOLE
                    ;16468  WAIT.T25.0:
U 13C9, 093C,94     ;16469          JMP [WAIT.T25.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                    ;16470                                           ; RESPOND
                    ;16471
U 13CA, 0A1A,9C     ;16472          JSR [SETUP]                      ;SET UP MASKS, MODULE CODE ETC. AND
                    ;16473                                           ; CLEAR MCT CSR 1
                    ;16474
U 13CB, B670,15     ;16475          MOV LS[OTHER.DATA] TO WR[0]      ;SET OTHER DATA PRINTOUT BIT
U 13CC, 3E80,15     ;16476          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL WORD TO PRINT ADDRESS
                    ;16477                                           ; UNDER 'OTHER' IF ERROR
U 13CD, 6512,15     ;16478          CLR LS[T9]                       ;ADDRESS TO WRITE IN MAIN MEMORY
U 13CE, 6588,15     ;16479          CLR LS[ADDRESS.DATA]             ;ALSO CLEAR ADDRESS TO PRINT
U 13CF, B62C,15     ;16480          MOV LS[#FFFFFF00] TO WR[0]       ;EXPECTED DATA (BYTE 0 CLEAR)
                    ;16481  REPEAT.T25.1:
U 13D0, BE14,15     ;16482          MOV WR[0] TO LS[T10]              ;STORE EXPECTED DATA IN LS
U 13D1, 999C,75     ;16483          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                    ;16484                                           ;SET UP TO WRITE LOCATION 0 OF MAIN MEM
U 13D2, 329E,15     ;16485          WRITE.MEM LS[ONES]               ;WRITE ONES IN LONGWORD AT 0
                    ;16486  LOOP.T25.1:
U 13D3, B614,95     ;16487          MOV LS[T10] TO WR[1]             ;EXPECTED DATA (ONE BYTE CLEAR)
U 13D4, 9912,15     ;16488          MEM.REQ[WRITE.P] ADRS[T9] DT[BYTE]
                    ;16489                                           ;SET UP TO WRITE A BYTE AT ADDRESS 0
U 13D5, B29C,15     ;16490          WRITE.MEM LS[ZERO]               ;WRITE ZERO IN BYTE SPECIFIED IN LS 9
U 13D6, 189D,F5     ;16491          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                    ;16492                                           ;SET UP TO READ LONGWORD AT ADDRESS 0
U 13D7, 3022,15     ;16493          MOV MEM.DATA TO WR[0]            ;GET LONGWORD
U 13D8, 0869,3C     ;16494          JSR [CHECK.RESULT]               ;CHECK FOR ONE BYTE 0 = 0, OTHER BYTES=1'S
U 13D9, 893D,34     ;16495          JMP [LOOP.T25.1]                 ;ERROR LOOP IF ENABLED
U 13DA, 0944,1C     ;16496          JSR [SHIFT.T25.8BITS]            ;PREPARE FOR NEXT BYTE POSITION
U 13DB, B644,95     ;16497          MOV LS[#4] TO WR[1]              ;PUT A 4 IN WR
                    ;16498          CMP WR[1] WITH LS[T9],           ;SEE IF BYTE 3 WAS TESTED
U 13DC, CF12,B5     ;16499           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 13DD, 893D,01     ;16500          JMP.IF[NEQ] TO [REPEAT.T25.1]    ;REPEAT UNTIL BYTES 0-3 HAVE BEEN TESTED
                    ;16501
U 13DE, FF82,15     ;16502          INC LS[ERROR.NUMBER]             ;ERROR 2
U 13DF, 3683,95     ;16503          MOV LS[ERROR.NUMBER] TO WR[3]    ;SAVE ERROR NUMBER
U 13E0, 6512,15     ;16504          CLR LS[T9]                       ;ADDRESS TO WRITE IN MAIN MEMORY
U 13E1, 5F28,15     ;16505          MCOM LS[#FFFF] TO WR[0]          ;EXPECTED DATA (BYTES 0 AND 1 CLEAR)
                    ;16506  REPEAT.T25.2:
```

```
U 13E2, BE14,15 ;16507          MOV WR[0] TO LS[T10]             ;STORE EXPECTED DATA IN LS
U 13E3, 999C,75 ;16508          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;16509                                           ;SET UP TO WRITE LOCATION 0 OF MAIN MEM
U 13E4, 329E,15 ;16510          WRITE.MEM LS[ONES]               ;WRITE ONES IN LONGWORD AT 0
                ;16511  LOOP.T25.2:
U 13E5, B614,95 ;16512          MOV LS[T10] TO WR[1]             ;EXPECTED DATA (ONE WORD CLEAR)
U 13E6, 1912,35 ;16513          MEM.REQ[WRITE.P] ADRS[T9] DT[WORD]
                ;16514                                           ;SET UP TO WRITE A WORD AT ADDRESS 0
U 13E7, B29C,15 ;16515          WRITE.MEM LS[ZERO]               ;WRITE ZEROS IN WORD SPECIFIED IN LS 9
U 13E8, 189D,F5 ;16516          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;16517                                           ;SET UP TO READ LONGWORD AT ADDRESS 0
U 13E9, 3022,15 ;16518          MOV MEM.DATA TO WR[0]            ;GET LONGWORD
U 13EA, 0869,3C ;16519          JSR [CHECK.RESULT]               ;CHECK FOR ONE WORD = 0, OTHER WORD=1'S
U 13EB, 893E,54 ;16520          JMP [LOOP.T25.2]                 ;ERROR LOOP IF ENABLED
                ;16521
U 13EC, 0944,1C ;16522          JSR [SHIFT.T25.8BITS]            ;PREPARE FOR NEXT BYTE POSITION
U 13ED, B6C0,95 ;16523          MOV LS[#3(H)] TO WR[1]           ;PUT A 3 IN WR
                ;16524          CMP WR[1] WITH LS[T9],           ;SEE IF DONE WITH WORD STARTING AT BYTE
U 13EE, CF12,B5 ;16525           DT(LONG)&SET.ALU.CC             ; POSITION 2 AND SET CONDITION CODES
U 13EF, 093E,21 ;16526          JMP.IF[NEQ] TO [REPEAT.T25.2]    ;DONE CHECKING SINGLE CYCLE WRITES.
                ;16527                                           ; CHECK TWO CYCLE WRITE
                ;16528
U 13F0, 999C,75 ;16529          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;16530                                           ;SET UP TO WRITE LOCATION 0 OF MAIN MEM
U 13F1, 329E,15 ;16531          WRITE.MEM LS[ONES]               ;WRITE ONES IN LONGWORD AT 0
U 13F2, 9944,75 ;16532          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                ;16533                                           ;SET UP TO WRITE LOCATION 4 OF MAIN MEM
U 13F3, 329E,15 ;16534          WRITE.MEM LS[ONES]               ;WRITE ONES IN LONGWORD AT 4
                ;16535  LOOP.T25.3:
U 13F4, BE83,95 ;16536          MOV WR[3] TO LS[ERROR.NUMBER]    ;ERROR 2
U 13F5, 6588,15 ;16537          CLR LS[ADDRESS.DATA]             ;ADDRESS TO PRINT IF ERROR
                ;16538  LOOP.T25.2B:
U 13F6, 5F2A,95 ;16539          MCOM LS[#FF000000] TO WR[1]      ;EXPECTED DATA (0'S IN BYTE 3)
U 13F7, 1912,35 ;16540          MEM.REQ[WRITE.P] ADRS[T9] DT[WORD]
                ;16541                                           ;SET UP TO WRITE A WORD AT ADDRESS 0
U 13F8, B29C,15 ;16542          WRITE.MEM LS[ZERO]               ;WRITE ZEROS STARTING AT BYTE 3
U 13F9, 189D,F5 ;16543          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;16544                                           ;SET UP TO READ LONGWORD AT ADDRESS 0
U 13FA, 3022,15 ;16545          MOV MEM.DATA TO WR[0]            ;GET LONGWORD
U 13FB, 0869,3C ;16546          JSR [CHECK.RESULT]               ;CHECK FOR BYTE 3=0'S
U 13FC, 093F,64 ;16547          JMP [LOOP.T25.2B]                ;ERROR LOOP IF ENABLED
                ;16548
U 13FD, FF82,15 ;16549          INC LS[ERROR.NUMBER]             ;ERROR 3
U 13FE, 3644,15 ;16550          MOV LS[#4] TO WR[0]              ;PUT A 4 IN WR
U 13FF, BE88,15 ;16551          MOV WR[0] TO LS[ADDRESS.DATA]    ;ADDRESS TO PRINT IF ERROR (ADDRESS 4)
                ;16552
U 1400, 362C,95 ;16553          MOV LS[#FFFFFF00] TO WR[1]       ;EXPECTED DATA (BYTE 0 CLEAR IN LOCATION 4
U 1401, 1845,F5 ;16554          MEM.REQ[READ.P] ADRS[#4] DT[LONG]
                ;16555                                           ;SET UP TO READ LONGWORD AT ADDRESS 4
U 1402, 3022,15 ;16556          MOV MEM.DATA TO WR[0]            ;GET LONGWORD
U 1403, 0869,3C ;16557          JSR [CHECK.RESULT]               ;CHECK FOR BYTE 0=0'S IN LONGWORD 4
U 1404, 893F,44 ;16558          JMP [LOOP.T25.3]                 ;ERROR LOOP IF ENABLED
                ;16559
U 1405, FF82,15 ;16560          INC LS[ERROR.NUMBER]             ;ERROR 4
U 1406, 6588,15 ;16561          CLR LS[ADDRESS.DATA]             ;ADDRESS TO PRINT IF ERROR
```

```
U 1407, 999C,75 ;16562          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;16563                                  ;SET UP TO WRITE IN LONGWORD 0
U 1408, 329E,15 ;16564          WRITE.MEM LS[ONES]      ;WRITE LOCATION 0 WITH LONGWORD OF 1'S
                ;16565  LOOP.T25.4:
U 1409, B69C,95 ;16566          MOV LS[ZERO] TO WR[1]   ;EXPECTED DATA (ALL 0'S)
U 140A, 999C,75 ;16567          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;16568                                  ;SET UP TO WRITE 0'S IN LONGWORD 0
U 140B, 0A1B,4C ;16569          JSR [NOP.DELAY]         ;EXECUTE 5 CYCLE DELAY
U 140C, DB00,15 ;16570          NOP                     ;DELAY ONE MOR CYCLE
U 140D, B29C,15 ;16571          WRITE.MEM LS[ZERO]      ;WRITE DATA AFTER DELAY OF 6 NOP'S
U 140E, 189D,F5 ;16572          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;16573                                  ;SET UP TO READ RESULT
U 140F, 3022,15 ;16574          MOV MEM.DATA TO WR[0]   ;GET RESULT OF LONGWORD 0
U 1410, 0869,3C ;16575          JSR [CHECK.RESULT]      ;CHECK FOR ALL 0'S
U 1411, 8940,94 ;16576          JMP [LOOP.T25.4]        ;LOOP ON ERROR IF ENABLED
                ;16577
U 1412, FF82,15 ;16578          INC LS[ERROR.NUMBER]    ;ERROR 5
U 1413, 3683,95 ;16579          MOV LS[ERROR.NUMBER] TO WR[3]   ;SAVE IN WR 3
U 1414, 6512,15 ;16580          CLR LS[T9]              ;ADDRESS TO WRITE IN MAIN MEMORY
U 1415, FF12,15 ;16581          INC LS[T9]              ;FIRST ADDRESS IS 1
U 1416, DF2C,15 ;16582          MCOM LS[#FFFFFF00] TO WR[0]     ;FIRST EXPECTED DATA (0'S IN BYTES
                ;16583                                  ; 1,2, AND 3) OF ADDRESS 0
                ;16584  REPEAT.T25.5:
U 1417, BE14,15 ;16585          MOV WR[0] TO LS[T10]    ;EXPECTED DATA FOR LOCATION 0
U 1418, F816,15 ;16586          MCOM WR[0] TO LS[T11]   ;EXPECTED DATA FOR LOCATION 4 (COMPLEMENT
                ;16587                                  ; OF EXP DATA FOR LOCATION 0)
U 1419, 999C,75 ;16588          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;16589                                  ;SET UP TO WRITE LOCATION 0 OF MAIN MEM
U 141A, 329E,15 ;16590          WRITE.MEM LS[ONES]      ;WRITE ONES IN LONGWORD AT 0
U 141B, 9944,75 ;16591          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                ;16592                                  ;SET UP TO WRITE LOCATION 4 OF MAIN MEM
U 141C, 329E,15 ;16593          WRITE.MEM LS[ONES]      ;WRITE ONES IN LONGWORD AT 4
                ;16594  LOOP.T25.6:
U 141D, BE83,95 ;16595          MOV WR[3] TO LS[ERROR.NUMBER]   ;ERROR 5
U 141E, 6588,15 ;16596          CLR LS[ADDRESS.DATA]    ;ADDRESS TO PRINT IF ERROR
                ;16597  LOOP.T25.5:
U 141F, B614,95 ;16598          MOV LS[T10] TO WR[1]    ;EXPECTED DATA
U 1420, 9912,75 ;16599          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;16600                                  ;SET UP TO WRITE A LONGWORD
U 1421, B29C,15 ;16601          WRITE.MEM LS[ZERO]      ;WRITE ZEROS STARTING AT BYTE POSITION
                ;16602                                  ; SPECIFIED IN LS 9
U 1422, 189D,F5 ;16603          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;16604                                  ;SET UP TO READ LONGWORD AT ADDRESS 0
U 1423, 3022,15 ;16605          MOV MEM.DATA TO WR[0]   ;GET LONGWORD
U 1424, 0869,3C ;16606          JSR [CHECK.RESULT]      ;CHECK FOR CORRECT RESULT
U 1425, 0941,F4 ;16607          JMP [LOOP.T25.5]        ;ERROR LOOP IF ENABLED
                ;16608
U 1426, FF82,15 ;16609          INC LS[ERROR.NUMBER]    ;ERROR 6
U 1427, 3644,15 ;16610          MOV LS[#4] TO WR[0]     ;PUT A 4 IN WR
U 1428, BE88,15 ;16611          MOV WR[0] TO LS[ADDRESS.DATA]   ;ADDRESS TO PRINT IF ERROR (ADDRESS 4)
                ;16612
U 1429, 3616,95 ;16613          MOV LS[T11] TO WR[1]    ;EXPECTED DATA FOR ADDRESS 4
U 142A, 1845,F5 ;16614          MEM.REQ[READ.P] ADRS[#4] DT[LONG]
                ;16615                                  ;SET UP TO READ LONGWORD AT ADDRESS 4
U 142B, 3022,15 ;16616          MOV MEM.DATA TO WR[0]   ;GET LONGWORD
```

```
U 142C, 0869,3C ;16617          JSR [CHECK.RESULT]                ;CHECK FOR CORRECT RESULT
U 142D, 8941,D4 ;16618          JMP [LOOP.T25.6]                  ;ERROR LOOP IF ENABLED
                ;16619
U 142E, B62A,15 ;16620          MOV LS[#FF000000] TO WR[0]        ;SET BITS 23-31 IN WR
U 142F, ED14,15 ;16621          BIS WR[0] TO LS[T10]              ;SET UPPER BYTE OF EXPECTED DATA.  1'S
                ;16622                                            ; WILL THEN BE ROTATED INTO BOTTOM BYTE
                ;16623                                            ; DURING SHIFT
U 1430, 0944,1C ;16624          JSR [SHIFT.T25.8BITS]             ;SET UP FOR NEXT BYTE POSITION
U 1431, B644,95 ;16625          MOV LS[#4] TO WR[1]               ;PUT A 4 IN WR 1
                ;16626          CMP WR[1] WITH LS[T9],            ;SEE IF DONE BYTE POSITION 3 YET
U 1432, CF12,B5 ;16627           DT(LONG)&SET.ALU.CC              ; AND SET CONDITION CODES
U 1433, 8941,71 ;16628          JMP.IF[NEQ] TO [REPEAT.T25.5]     ;REPEAT IF NOT DONE
                ;16629
U 1434, FF82,15 ;16630          INC LS[ERROR.NUMBER]              ;ERROR 7
U 1435, 6588,15 ;16631          CLR LS[ADDRESS.DATA]              ;CLEAR ADDRESS TO PRINT
U 1436, 999C,75 ;16632          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;16633                                            ;SET UP TO WRITE LOCATION 0 OF MAIN MEM
U 1437, 329E,15 ;16634          WRITE.MEM LS[ONES]                ;WRITE ONES IN LONGWORD AT 0
                ;16635  LOOP.T25.7:
U 1438, 362C,95 ;16636          MOV LS[#FFFFFF00] TO WR[1]        ;EXPECTED DATA (BYTE 0 CLEAR)
U 1439, 999C,15 ;16637          MEM.REQ[WRITE.P] ADRS[#0] DT[BYTE]
                ;16638                                            ;SET UP TO WRITE A BYTE AT ADDRESS 0
U 143A, 0A1B,4C ;16639          JSR [NOP.DELAY]                   ;EXECUTE 5 CYCLE DELAY
U 143B, B29C,15 ;16640          WRITE.MEM LS[ZERO]                ;WRITE ZERO IN BYTE 0
U 143C, 189D,F5 ;16641          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;16642                                            ;SET UP TO READ LONGWORD AT ADDRESS 0
U 143D, 3022,15 ;16643          MOV MEM.DATA TO WR[0]             ;GET LONGWORD
U 143E, 0869,3C ;16644          JSR [CHECK.RESULT]                ;CHECK FOR ONE BYTE 0 = 0, OTHER BYTES=1'S
U 143F, 0943,84 ;16645          JMP [LOOP.T25.7]                  ;ERROR LOOP IF ENABLED
U 1440, 8944,84 ;16646          JMP [END.T25]                     ;ELSE DONE WITH TEST
                ;16647
                ;16648
                ;16649  SHIFT.T25.8BITS:
U 1441, 3614,15 ;16650          MOV LS[T10] TO WR[0]              ;GET LAST EXPECTED DATA
U 1442, FF12,15 ;16651          INC LS[T9]                        ;NEXT BYTE POSITION
U 1443, 3646,95 ;16652          MOV LS[#8] TO WR[1]               ;SHIFT COUNTER
                ;16653  SHIFT.T25.LOOP:
U 1444, A3C0,15 ;16654          ROL WR[0]                         ;SHIFT EXPECTED DATA
                ;16655          DEC WR[1],                        ;DECREMENT COUNTER
U 1445, A102,B5 ;16656           DT(LONG)&SET.ALU.CC              ; AND SET CONDITION CODES
U 1446, 8944,41 ;16657          JMP.IF[NEQ] TO [SHIFT.T25.LOOP]   ;REPEAT UNTIL 8 SHIFTS COMPLETED
U 1447, 5B00,14 ;16658          RETURN                            ;DONE WITH SHIFT
                ;16659
                ;16660
                ;16661  END.T25:
```

```
;16662  .PAGE   "  TEST 26 MME Test (M8391  MCT Module)"
;16663  ;++
;16664  ;
;16665  ; Test Description:
;16666  ;
;16667  ;       This test checks the MISC CONTROL PAL and its input L MME H.  MME (Mem-
;16668  ;       ory Managment Enable) comes from CSR 1 and has already been tested at
;16669  ;       the CSR.  If MME is clear, the MISC CONTROL PAL will assert ADDR PH on
;16670  ;       a memory referance as long as the TB is not being written.  This test
;16671  ;       will keep this bit set while it writes the TB with a virtual address.
;16672  ;       Complement memory data is written into two locations in memory, one of
;16673  ;       which is pointed to by the TB, the other is not.  A READ.V.NOCHK is
;16674  ;       executed which should access the location pointed to by the TB.  Then
;16675  ;       CSR 1 is written to clear MME and the READ.V.NOCHK is repeated.  This
;16676  ;       time, the MISC CONTROL PAL should assert ADDR PH L and the physical
;16677  ;       location should be accessed.
;16678  ;         The last part of this test checks that even if memory management is
;16679  ;       disabled, that a WRITE or READ of the TB will still work (ADDR PH L is
;16680  ;       not asserted if the TB is accessed).
;16681  ;
;16682  ; Assumptions:
;16683  ;
;16684  ;       It is assumed that all previous tests have run successfully.
;16685  ;
;16686  ; Test Steps:
;16687  ;
;16688  ;       1)  Set up error mask, error number, and module number in LS (for
;16689  ;           error printouts) and clear previous error number in LS.
;16690  ;       2)  Load CSR to set the MME bit.
;16691  ;       3)  Load the TB with data of a 1 in the VALID bit, byte offset, and in
;16692  ;           PA 09 (bits 0 and 31 in data sent over) at TB address 0.  This
;16693  ;           makes a translation to physical address 200(H) (the second page of
;16694  ;           memory).
;16695  ;       4)  Write data of all 0's to physical address 0.  Write data of all
;16696  ;           ones to physical address 200(H).
;16697  ;       5)  Execute MEM.REQ with MF=READ.V.NOCHK using 0 as the address (trans-
;16698  ;           lates to ph addr 200(H)).  Check result for all ones.
;16699  ;       6)  Clear CSR 1 to disable memory management.
;16700  ;       7)  Repeat step 5 except check result for all zeros (should go to phy-
;16701  ;           sical address 0).
;16702  ;       8)  Change error mask to check bits 1 through 22.  Keeping memory man-
;16703  ;           agement disabled, write and read first all ones and then all zeros
;16704  ;           into TB address 0.  This checks that TB DATA FN L will disable
;16705  ;           ADDR PH.
;16706  ;
;16707  ; Errors:
;16708  ;
;16709  ;       Error 1 - Read of virtual address through the TB failed.  Should have
;16710  ;                 read physical address 200.  Expected and received data is
;16711  ;                 array data.
;16712  ;       Error 2 - Virtual READ with MME clear failed to go to physical address.
;16713  ;       Error 3 - TB access failed to force ADDR PH L high with address data of
;16714  ;                 of all ones.  Expected and received data is TB contents at TB
;16715  ;                 address 0.
;16716  ;       Error 4 - TB access failed to force ADDR PH L high with address data of
```

```
                ;16717  ;               of all zeros.  Expected and received data is TB contencs at TB
                ;16718  ;               address 0.
                ;16719  ;
                ;16720  ; Debug:
                ;16721  ;
                ;16722  ;       Two methods of debug will be explained.  If the MCT module is in a
                ;16723  ;       test station then the MCT can be single stepped making debug easier
                ;16724  ;       in some cases.  Otherwise only the CPU can be single stepped and this
                ;16725  ;       method will be explained also.  When single stepping the memory con-
                ;16726  ;       troller will help in debug, this section will explicitly suggest it.
                ;16727  ;
                ;16728  ;       Error 1 - This error indicate a problem with logic tested previously.
                ;16729  ;       Run the previous tests again.
                ;16730  ;
                ;16731  ;       Error 2 - This error indicates a problem with the MISC CONTROL PAL or
                ;16732  ;       its input L MME H.  The inputs L MME H can be checked after single
                ;16733  ;       stepping the CPU to the MEM.REQ instruction after the CSR is cleared.
                ;16734  ;       It should be low at the MISC CONTROL PAL.  If this is correct, the PAL
                ;16735  ;       is probably bad (the other inputs have already been tested).
                ;16736  ;
                ;16737  ;       Error 3 and 4 - This error indicates a problem with the MISC CONTROL PAL or
                ;16738  ;       the input TB DATA EN L.  Check TB DATA EN L while looping on error.  It
                ;16739  ;       should go low during the time that the TB is accessed.  If this is OK,
                ;16740  ;       the MISC CONTROL PAL is probably bad.
                ;16741  ;--
                ;16742
                ;16743  T.26:
                ;16744
U 1448, B65E,15 ;16745          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;16746                                          ; STATUS WORD
U 1449, 3E80,15 ;16747          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;16748                                          ; BIT 15 INDICATES START OF TEST TO
                ;16749                                          ; CONSOLE
U 144A, 10E0,15 ;16750          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;16751  WAIT.T26.0:
U 144B, 8944,B4 ;16752          JMP [WAIT.T26.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;16753                                          ; RESPOND
                ;16754
U 144C, 0A1A,AC ;16755          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                ;16756
U 144D, 0A17,DC ;16757          JSR [WRITE.CSR1.MME]            ;ENABLE MEMORY MANAGEMENT
                ;16758
U 144E, B640,15 ;16759          MOV LS[BIT0] TO WR[0]           ;SET BIT 0 (GOES TO PA 09 IN TB)
U 144F, 477E,15 ;16760          BIS LS[BIT31] TO WR[0]          ;SET BIT 31 (GOES TO VALID BIT IN TB)
U 1450, 4772,15 ;16761          BIS LS[BIT25] TO WR[0]          ;SET BIT 25 (GOES TO BYTE OFFSET IN TB)
U 1451, BE14,15 ;16762          MOV WR[0] TO LS[T10]            ;STORE DATA IN LS FOR WRITE
U 1452, 989C,F5 ;16763          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;16764                                          ;SET UP TO WRITE TB ADDRESS 0
U 1453, B214,15 ;16765          WRITE.MEM LS[T10]               ;SET VALID, BYTE OFFSET, AND PA 09 IN TB
U 1454, 999C,75 ;16766          MEM.REQ[WRITE.P] ADR  #0] DT[LONG]
                ;16767                                          ;SET UP TO WRITE AT ADDRESS 0
U 1455, B29C,15 ;16768          WRITE.MEM LS[ZERO]              ;WRITE DATA OF ALL ZEROS AT ADDRESS 0
U 1456, 1952,75 ;16769          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                ;16770                                          ;SET UP TO WRITE AT ADDRESS 200
U 1457, 329E,15 ;16771          WRITE.MEM LS[ONES]              ;WRITE DATA OF ALL ONES AT ADDRESS 200
```

```
                ;16772  LOOP.T26.1:
U 1458, 369E,95 ;16773          MOV LS[ONES] TO WR[1]            ;EXPECTED DATA
U 1459, 9A9C,75 ;16774          MEM.REQ[READ.V.NOCHK] ADRS[#0] DT[LONG]
                ;16775                                           ;SET UP TO READ AT VIRTUAL ADDRESS 0
U 145A, 3022,15 ;16776          MOV MEM.DATA TO WR[0]            ;GET DATA AT PHYSICAL ADDRESS 200
U 145B, 0869,3C ;16777          JSR [CHECK.RESULT]               ;CHECK FOR ALL 1'S
U 145C, 0945,84 ;16778          JMP [LOOP.T26.1]                 ;ERROR LOOP IF ENABLED
                ;16779
U 145D, FF82,15 ;16780          INC LS[ERROR.NUMBER]             ;ERROR 2
U 145E, 0A17,EC ;16781          JSR [CLEAR.CSR1]                 ;DISABLE MEMORY MANAGEMENT
                ;16782  LOOP.T26.2:
U 145F, 2F82,95 ;16783          CLR WR[1]                        ;EXPECTED DATA
U 1460, 9A9C,75 ;16784          MEM.REQ[READ.V.NOCHK] ADRS[#0] DT[LONG]
                ;16785                                           ;SET UP TO READ A PHYSICAL ADDRESS 0
U 1461, 3022,15 ;16786          MOV MEM.DATA TO WR[0]            ;GET DATA AT PHYSICAL ADDRESS 0 (MME
                ;16787                                           ; CLEAR FORCES ADDR PH)
U 1462, 0869,3C ;16788          JSR [CHECK.RESULT]               ;CHECK FOR DATA OF ALL 0'S
U 1463, 8945,F4 ;16789          JMP [LOOP.T26.2]                 ;ERROR LOOP IF ENABLED
                ;16790
U 1464, FF82,15 ;16791          INC LS[ERROR.NUMBER]             ;ERROR 3
U 1465, B62A,15 ;16792          MOV LS[#FF000000] TO WR[0]       ;SET TOP BYTE FOR ERROR MASK
U 1466, C76E,15 ;16793          BIS LS[BIT23] TO WR[0]           ;SET BIT 23
U 1467, C740,15 ;16794          BIS LS[BIT0] TO WR[0]            ;AND SET BIT 0.  NOW BITS 1 THROUGH 22
                ;16795                                           ; WILL BE CHECKED FOR ERRORS
U 1468, 3E8A,15 ;16796          MOV WR[0] TO LS[ERROR.MASK]      ;STORE IN LS ERROR MASK
                ;16797
                ;16798  LOOP.T26.3:
U 1469, 369E,95 ;16799          MOV LS[ONES] TO WR[1]            ;EXPECTED DATA
U 146A, 989C,F5 ;16800          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;16801                                           ;SET UP TO WRITE TB ADDRESS 0
U 146B, 329E,15 ;16802          WRITE.MEM LS[ONES]               ;WRITE ALL ONES IN TB
U 146C, 9C9D,F5 ;16803          MEM.REQ[READ.TB] ADRS[#0] DT[LONG]
                ;16804                                           ;SET UP TO READ TB
U 146D, 3022,15 ;16805          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 146E, 0869,3C ;16806          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 146F, 8946,94 ;16807          JMP [LOOP.T26.3]                 ;ERROR LOOP IF ENABLED
                ;16808
U 1470, FF82,15 ;16809          INC LS[ERROR.NUMBER]             ;ERROR 4
                ;16810  LOOP.T26.4:
U 1471, B69C,95 ;16811          MOV LS[ZERO] TO WR[1]            ;EXPECTED DATA
U 1472, 989C,F5 ;16812          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;16813                                           ;SET UP TO WRITE TB ADDRESS 0
U 1473, B29C,15 ;16814          WRITE.MEM LS[ZERO]               ;WRITE ALL ZEROS IN TB
U 1474, 9C9D,F5 ;16815          MEM.REQ[READ.TB] ADRS[#0] DT[LONG]
                ;16816                                           ;SET UP TO READ TB
U 1475, 3022,15 ;16817          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 1476, 0869,3C ;16818          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 1477, 8947,14 ;16819          JMP [LOOP.T26.4]                 ;ERROR LOOP IF ENABLED
                ;16820
                ;16821  END.T26:
```

```
;16822  .PAGE   "  TEST 27 Prefetch Logic (M8391  MCT or M8390 CPU  Module)"
;16823  ;++
;16824  ;
;16825  ; Test Description:
;16826  ;
;16827  ;       This test checks the Prefetch logic on the MCT module.  128 longwords
;16828  ;       (one page) are written into memory starting at physical address 0
;16829  ;       with data equal to its own address in the top byte (unique data).  Ad-
;16830  ;       dress 200 (the 129th longword) is written with alternating 1's and 0's.
;16831  ;       The test issues a MEM.REQ with MF=READ.V.RCHK.IFILL.  The MME bit has
;16832  ;       been cleared in CSR 1 previously (memory managemant is disabled) so
;16833  ;       that a physical address is used (this eliminates having to set up the
;16834  ;       TB). The READ.V.RCHK.IFILL reads the memory location sent over with the
;16835  ;       MEM.REQ and sends it to the CPU to be loaded into the IB PREFETCH reg-
;16836  ;       ister.  The PREFETCH ADDRESS COUNT PAL on the MCT module should have
;16837  ;       incremented at the end of this MEM.REQ cycle so that it is now point-
;16838  ;       ing to the next longword in memory. Now a DECODE instruction is issued
;16839  ;       with MOV IB.DATA TO OS and the OS is read for the first data pattern
;16840  ;       (a 0 corresponding to memory address 0).  This decode was executed with
;16841  ;       PC=3 (LS 10 was loaded with a 3) and should have initiated another mem-
;16842  ;       ory request using the incremented address from the PREFETCH ADDRESS
;16843  ;       COUNT PAL.  The MOV IB.DATA TO OS is executed again.  This time, the
;16844  ;       second address data should be seen in OS (a 1).  This verifies the LOAD
;16845  ;       IB logic and the enable on the PREFETCH ADDRESS COUNT PAL.  The rest of
;16846  ;       the test reads the remaining page to verify the PREFETCH ADDRESS COUNT
;16847  ;       PAL.  Finally one more MOV IB.DATA TO OS is executed to check that a
;16848  ;       PAGE BOUNDARY is generated.  The memory micro-code should increment the
;16849  ;       VAR and read the pattern at address 200(H).
;16850  ;
;16851  ; Assumptions:
;16852  ;
;16853  ;       It is assumed that all previous tests have run successfully.  This test
;16854  ;       uses CPU logic that may not have been checked yet.  The CPU diagnostic
;16855  ;       can not test the IB PREFETCH logic until the memory has been verified.
;16856  ;       The CPU micro-diagnostic that is run after memory verification, tests
;16857  ;       this logic again from the CPU's point of view.  If a known good CPU
;16858  ;       module is being used, then a failure in this test is a result of the
;16859  ;       MCT module.  The debug portion assumes that the CPU is good.
;16860  ;
;16861  ; Test Steps:
;16862  ;
;16863  ;       1)  Set up error mask, error number, and module number in LS (for
;16864  ;           error printouts), clear previous error number in LS, and clear
;16865  ;           MME in CSR 1.
;16866  ;       2)  Write a 0 in memory at physical address 0.  Write a 1 in byte 3
;16867  ;           of physical address 4 (data of 1000000(H)).  Write a 2 in byte 3
;16868  ;           of physical address 8 (data of 2000000(H)).  Continue until one
;16869  ;           page of memory (128 longwords) has been written.  Then write an
;16870  ;           alternating 1's and 0's pattern in address 200.
;16871  ;       3)  Load OS with all 1's and load LS 10 with a 3.  Issue a MEM.REQ with
;16872  ;           MF=READ.V.RCHK.IFILL using LS 10 as the source of the address.
;16873  ;       4)  Issue a MOV IB.DATA TO OS instruction.  This moves byte 3 of the
;16874  ;           prefetch register into OS.  Read OS for a 0.
;16875  ;       5)  Load LS 10 with 203(H) and repeat steps 3 and 4.  The result should
;16876  ;           be alternating 1's and 0's (55(H)).
```

```
;16877  ;       6)  Repeat steps 3 and 4, but do not check the result.  The MOV IB.DATA
;16878  ;           TO OS in step for should have loaded the IB with a next longword of
;16879  ;           data (address 4).  Increment the LS 10 (macro PC) by 2 so that it
;16880  ;           is now equal to 6.
;16881  ;       7)  Issue two MOV IB.DATA TO OS instructions.  The first sets up PC=3,
;16882  ;           the second reads the third byte of IB and does the next prefetch.
;16883  ;       8)  Check the OS for correct data.  Increment the macro PC by 2 and
;16884  ;           repeat steps 7 and 8 until the last longword on the page has been
;16885  ;           accessed (longword at address 1FC).
;16886  ;       9)  Issue two more decodes (MOV IB.DATA TO OS) and check OS for alter-
;16887  ;           nating 1's and 0's.  This checks the page boundary logic in the
;16888  ;           micro-flows.
;16889  ;
;16890  ; Errors:
;16891  ;
;16892  ;       NOTE:  Expected and received data is contents of OS which was loaded
;16893  ;              from IB reg (8 bits).
;16894  ;
;16895  ;       Error 1 - READ.V.RCHK.IFILL failed to load the IB correctly, or the MOV
;16896  ;                 IB.DATA TO OS failed in the CPU with data of 0's.
;16897  ;       Error 2 - READ.V.RCHK.IFILL failed to load the IB correctly, or the MOV
;16898  ;                 IB.DATA TO OS failed in the CPU with data of alternating 1's
;16899  ;                 and 0's.
;16900  ;       Error 3 - Decode instruction MOV IB.DATA TO OS failed to prefetch next
;16901  ;                 longword location.
;16902  ;       Error 4 - PREFETCH ADDRESS COUNT PAL failed to increment correctly.
;16903  ;       Error 5 - PAGE BOUNDARY logic failed.
;16904  ;
;16905  ; Debug:
;16906  ;
;16907  ;       Two methods of debug will be explained.  If the MCT module is in a
;16908  ;       test station then the MCT can be single stepped making debug easier
;16909  ;       in some cases.  Otherwise only the CPU can be single stepped and this
;16910  ;       method will be explained also.  When single stepping the memory con-
;16911  ;       troller will help in debug, this section will explicitly suggest it.
;16912  ;
;16913  ;       NOTE:  If the memory controller hangs (causing the CPU to timeout on
;16914  ;       the MOV IB.DATA TO OS instruction) the problem is probably on the CPU
;16915  ;       module.  The CPU is responsible for asserting DATA RCVD on the decode
;16916  ;       instruction.  The memory will hang in a loop looking for CPU grant to
;16917  ;       go away if DATA RCVD is not asserted by the CPU.
;16918  ;
;16919  ;       Errors 1 and 2 - This error indicates a problem with the prefetch logic
;16920  ;       on either the MCT module or the CPU module. It is assumed in this debug
;16921  ;       procedure that the CPU module is working.
;16922  ;         While looping on error, check the signal LOAD IB H going out of the
;16923  ;       memory controller module.  If this signal does not pulse high in the
;16924  ;       error loop, check the signal I LD IB L (C36) going into the negated
;16925  ;       input 'AND' gate.  This signal comes directly from the memory control
;16926  ;       store.  Also check the ENABLE ERROR H goes low during the loop.
;16927  ;         If LOAD IB H is getting off the MCT module, the problem is probably
;16928  ;       on the CPU module.
;16929  ;
;16930  ;       Error 3 - This error indicates a problem with the prefetch logic on
;16931  ;       either the MCT module or the CPU module.  It is assumed in this debug
```

```
                ;16932  ;       procedure that the CPU module is working.
                ;16933  ;         While looping on error, check the signal LOAD IB H going into the
                ;16934  ;       PREFETCH ADDRESS COUNT PAL.  This signal should be pulsing high during
                ;16935  ;       the error loop.  Trace it back if it is not going high.
                ;16936  ;         If LOAD IB H is pulsing, check the signal OP PREF ADR L.  It should
                ;16937  ;       be low for most of the memory cycle.  If not, check it at the CONTROL
                ;16938  ;       PREFETCH PAL.  If incorrect there, check the inputs to the PAL for a
                ;16939  ;       low on CSR 19 H at the time that CONT FUNC LAT L is high and CPU GRANT
                ;16940  ;       L is low.  If these inputs are OK, the PAL is probably bad.
                ;16941  ;         If OP PREF ADR L is going low, make sure that OP ARY ADR L is going
                ;16942  ;       high at the same time as OP PREF ADR L is low.  If it is, the problem
                ;16943  ;       is probably in the CPU module.
                ;16944  ;
                ;16945  ;       Error 4 - Suspect the PREFETCH ADDRESS COUNT PAL itself, or no connec-
                ;16946  ;       tion or one of the LVA 0x H inputs.
                ;16947  ;
                ;16948  ;       Error 5 - While looping on error, check the signal PG BND PREF H.  It
                ;16949  ;       should be pulsing high.  If it is, check it going into the CONTROL PRE-
                ;16950  ;       FETCH PAL.
                ;16951  ;         If PG BND PREF H is OK at the CONTROL PREFETCH PAL, the PAL is prob-
                ;16952  ;       ably bad.
                ;16953  ;
                ;16954  ;--
                ;16955
                ;16956  T.27:
                ;16957
U 1478, B65E,15 ;16958          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;16959                                          ; STATUS WORD
U 1479, 3E80,15 ;16960          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;16961                                          ; BIT 15 INDICATES START OF TEST TO
                ;16962                                          ; CONSOLE
U 147A, 10E0,15 ;16963          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;16964  WAIT.T27.0:
U 147B, 8947,B4 ;16965          JMP [WAIT.T27.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;16966                                          ; RESPOND
                ;16967
U 147C, 0A1A,9C ;16968          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                ;16969                                          ; CLEAR MCT CSR 1
U 147D, 4742,15 ;16970          BIS LS[CPU] TO WR[0]            ;ALSO SET BIT FOR CPU
U 147E, 3E8C,15 ;16971          MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                ;16972                                          ; MCT OR CPU BOARD
U 147F, B62C,15 ;16973          MOV LS[#FFFFFF00] TO WR[0]      ;BITS 0-7=0, BITS 8-31=1'S
U 1480, 3E8A,15 ;16974          MOV WR[0] TO LS[ERROR.MASK]     ;SET UP ERROR MASK TO CHECK 8 BITS
                ;16975
U 1481, 3653,15 ;16976          MOV LS[#200] TO WR[2]           ;DECIMAL 128X4 IN WR
U 1482, B671,95 ;16977          MOV LS[BIT24] TO WR[3]          ;PUT 1000000(H) IN WR 3 TO INCREMENT
                ;16978                                          ; DATA PATTERN
U 1483, 6512,15 ;16979          CLR LS[T9]                      ;ADDRESS TO WRITE IN MAIN MEMORY
U 1484, 6514,15 ;16980          CLR LS[T10]                     ;DATA TO WRITE AT ADDRESS IN LS 9
                ;16981  BACK.T27:
U 1485, 9912,75 ;16982          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;16983                                          ;SET UP TO WRITE IN MEMORY
U 1486, B214,15 ;16984          WRITE.MEM LS[T10]               ;WITH DATA FROM LS T10 (DATA=ADDRESS IN
                ;16985                                          ; BYTE 3 OF LONGWORD)
U 1487, 3644,15 ;16986          MOV LS[#4] TO WR[0]             ;PUT A 4 IN WR
```

```
U 1488, 6C12,15 ;16987          ADD WR[0] TO LS[T9]                ;NEXT ADDRESS                                  U
U 1489, 6C15,95 ;16988          ADD WR[3] TO LS[T10]               ;ADD 1000000 TO DATA (INCREMENT BYTE 3)
                ;16989          CMP LS[T9] WITH WR[2],            ;SEE IF DONE 128 LONGWORDS (ONE PAGE)
U 148A, 4E13,35 ;16990           DT(LONG)&SET.ALU.CC              ; AND SET CONDITION CODES                      U
U 148B, 0948,51 ;16991          JMP.IF[NEQ] TO [BACK.T27]         ;REPEAT UNTIL ONE FULL PAGE IS WRITTEN
U 148C, B69A,95 ;16992          MOV LS[#55555555] TO WR[1]        ;ALTERNATING 1'S AND 0'S TO WR 1               U
U 148D, DF2A,15 ;16993          MCOM LS[#FF000000] TO WR[0]       ;00FFFFFF IN WR 0
U 148E, AF40,95 ;16994          BIC WR[0] TO WR[1]                ;DATA PATTERN OF 55000000
U 148F, 3E14,95 ;16995          MOV WR[1] TO LS[T10]              ;PUT DATA IN LS FOR WRITE                      U
U 1490, 9912,75 ;16996          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;16997                                            ;SET UP TO WRITE IN MEMORY ADDR 200            U
U 1491, B214,15 ;16998          WRITE.MEM LS[T10]                 ;WITH DATA FROM LS T10 (ALT 1'S & 0'S)
                ;16999
U 1492, B69E,15 ;17000          MOV LS[ONES] TO WR[0]             ;PUT ALL ONES IN WR 0                          U
U 1493, 3EF8,15 ;17001          MOV WR[0] TO LS[OS]               ;LOAD OS REG WITH ALL 1'S
                ;17002  LOOP.T27.1:
U 1494, 36C0,15 ;17003          MOV LS[#3(H)] TO WR[0]            ;PUT A 3 IN WR                                 U
U 1495, 3E20,15 ;17004          MOV WR[0] TO LS[PC]               ;LOAD LS LOCATION 10 (PC)                      U
U 1496, 1B21,75 ;17005          MEM.REQ[READ.V.RCHK.IFILL] ADRS[PC] DT[LONG]                                     U
                ;17006                                            ;LOAD PREFETCH REG FROM MEMORY
U 1497, 2F82,95 ;17007          CLR WR[1]                         ;EXPECTED DATA                                 U
U 1498, 0A1B,4C ;17008          JSR [NOP.DELAY]                   ;EXECUTE 5 CYCLE DELAY TO ALLOW IB FILL        U
                ;17009                                            ; TO FINISH
U 1499, 8401,4E ;17010          MOV IB.DATA TO OS                 ;LOAD CURRENT DATA FROM BYTE 3 OF PRE-         U
                ;17011                                            ; FETCH REG INTO OS
U 149A, DB00,15 ;17012          NOP                               ;SKIP IF IB VALID SHOULD SKIP THIS             U
U 149B, B6F8,15 ;17013          MOV LS[OS] TO WR[0]               ;GET RESULT INTO WR 0                          U
U 149C, 0869,3C ;17014          JSR [CHECK.RESULT]                ;CHECK OS FOR CORRECT DATA                     U
U 149D, 0949,44 ;17015          JMP [LOOP.T27.1]                  ;ERROR LOOP IF ENABLED
                ;17016                                                                                           U
U 149E, FF82,15 ;17017          INC LS[ERROR.NUMBER]              ;ERROR 2                                       U
                ;17018  LOOP.T27.2:                                                                              U
U 149F, B652,15 ;17019          MOV LS[#200] TO WR[0]             ;WR GETS 200                                   U
U 14A0, C0C0,15 ;17020          ADD LS[#3(H)] TO WR[0]            ;WR NOW HAS 203                                U
U 14A1, 3E20,15 ;17021          MOV WR[0] TO LS[PC]               ;SET UP PC                                     U
U 14A2, 1B21,75 ;17022          MEM.REQ[READ.V.RCHK.IFILL] ADRS[PC] DT[LONG]                                     U
                ;17023                                            ;LOAD PREFETCH REG FROM MEMORY
U 14A3, B69A,95 ;17024          MOV LS[#55555555] TO WR[1]        ;EXPECTED DATA                                 U
U 14A4, 0A1B,4C ;17025          JSR [NOP.DELAY]                   ;EXECUTE 5 CYCLE DELAY TO ALLOW IB FILL        U
                ;17026                                            ; TO FINISH
U 14A5, 8401,4E ;17027          MOV IB.DATA TO OS                 ;LOAD CURRENT DATA FROM BYTE 3 OF PRE-         U
                ;17028                                            ; FETCH REG INTO OS
U 14A6, DB00,15 ;17029          NOP                               ;SKIP IF IB VALID SHOULD SKIP THIS             U
U 14A7, B6F8,15 ;17030          MOV LS[OS] TO WR[0]               ;GET RESULT INTO WR 0                          U
U 14A8, 0869,3C ;17031          JSR [CHECK.RESULT]                ;CHECK OS FOR CORRECT DATA                     U
U 14A9, 8949,F4 ;17032          JMP [LOOP.T27.2]                  ;ERROR LOOP IF ENABLED
                ;17033                                                                                           U
U 14AA, FF82,15 ;17034          INC LS[ERROR.NUMBER]              ;ERROR 3                                       U
U 14AB, 3641,15 ;17035          MOV LS[#1] TO WR[2]               ;INIT WR 2 FOR EXPECTED RESULT
U 14AC, 36C1,95 ;17036          MOV LS[#3(H)] TO WR[3]            ;WR 3 CONTAINS 3
                ;17037  LOOP.T27.3.4:                                                                            U
U 14AD, 3E21,95 ;17038          MOV WR[3] TO LS[PC]               ;SET UP PC                                     U
U 14AE, 1B21,75 ;17039          MEM.REQ[READ.V.RCHK.IFILL] ADRS[PC] DT[LONG]                                     U
                ;17040                                            ;INIT PREFETCH REG AND SET PC EQUALS 3         U
                ;17041                                            ; ON CPU MODULE                                U
```

```
U 14AF, 0A1B,4C ;17042          JSR [NOP.DELAY]                    ;EXECUTE 5 CYCLE DELAY TO ALLOW IB FILL
                ;17043                                             ; TO FINISH
U 14B0, 8401,4E ;17044          MOV IB.DATA TO OS                  ;DO A PREFETCH AND LOAD CURRENT DATA
                ;17045                                             ; FROM BYTE 3 OF PREFETCH REG INTO OS
U 14B1, DB00,15 ;17046          NOP                                ;SKIP IF IB VALID SHOULD SKIP THIS
U 14B2, 0A1B,4C ;17047          JSR [NOP.DELAY]                    ;EXECUTE 5 CYCLE DELAY TO ALLOW PRE-
                ;17048                                             ; FETCH TO FINISH
U 14B3, 3642,15 ;17049          MOV LS[#2] TO WR[0]                ;2 IN WR 0
U 14B4, EC20,15 ;17050          ADD WR[0] TO LS[PC]                ;INC PC BY 2
                ;17051  REPEAT.T27.3.4:
U 14B5, A004,95 ;17052          MOV WR[2] TO WR[1]                 ;EXPECTED DATA
U 14B6, 8401,4E ;17053          MOV IB.DATA TO OS                  ;SET PC EQUALS THREE AND READ A BYTE
U 14B7, DB00,15 ;17054          NOP                                ;SKIP IF IB VALID SHOULD SKIP THIS
U 14B8, 8401,4E ;17055          MOV IB.DATA TO OS                  ;DO A PREFETCH AND LOAD CURRENT DATA
                ;17056                                             ; FROM BYTE 3 OF PREFETCH REG INTO OS
U 14B9, DB00,15 ;17057          NOP                                ;SKIP IF IB VALID SHOULD SKIP THIS
U 14BA, B6F8,15 ;17058          MOV LS[OS] TO WR[0]                ;GET RESULT INTO WR 0
U 14BB, 0869,3C ;17059          JSR [CHECK.RESULT]                 ;CHECK OS FOR CORRECT DATA
U 14BC, 094A,D4 ;17060          JMP [LOOP.T27.3.4]                 ;ERROR LOOP IF ENABLED
                ;17061
U 14BD, 3644,15 ;17062          MOV LS[#4] TO WR[0]                ;PUT A 4 IN WR
U 14BE, BE82,15 ;17063          MOV WR[0] TO LS[ERROR.NUMBER]      ;THIS LOOP IS NOW ERROR 4
U 14BF, 3642,15 ;17064          MOV LS[#2] TO WR[0]                ;GOING TO ADD 2
U 14C0, EC20,15 ;17065          ADD WR[0] TO LS[PC]                ;INCREMENT MACRO PC BY 2
U 14C1, C045,95 ;17066          ADD LS[#4] TO WR[3]                ;INC OLD PC (FOR ERROR LOOPING)
U 14C2, 2045,15 ;17067          INC WR[2]                          ;INCREMENT EXPECTED RESULT
                ;17068          CMP WR[2] WITH LS[#80],            ;SEE IF PAGE BOUNDARY IS NEXT
U 14C3, CF4F,35 ;17069           DT(LONG)&SET.ALU.CC               ; AND SET CONDITION CODES
U 14C4, 094B,51 ;17070          JMP.IF[NEQ] TO [REPEAT.T27.3.4]    ;REPEAT UNTIL ONE PAGE TESTED
                ;17071
U 14C5, FF82,15 ;17072          INC LS[ERROR.NUMBER]               ;ERROR 5
                ;17073  LOOP.T27.5:
U 14C6, B69A,95 ;17074          MOV LS[#55555555] TO WR[1]         ;EXPECTED DATA
U 14C7, 8401,4E ;17075          MOV IB.DATA TO OS                  ;SET UP PC=3 FOR NEXT DECODE
U 14C8, DB00,15 ;17076          NOP                                ;SKIP IF IB VALID SHOULD SKIP THIS
U 14C9, 8401,4E ;17077          MOV IB.DATA TO OS                  ;DO A PREFETCH AND LOAD CURRENT DATA
                ;17078                                             ; FROM BYTE 3 OF PREFETCH REG INTO OS
U 14CA, DB00,15 ;17079          NOP                                ;SKIP IF IB VALID SHOULD SKIP THIS
U 14CB, B6F8,15 ;17080          MOV LS[OS] TO WR[0]                ;GET RESULT INTO WR 0
U 14CC, 0869,3C ;17081          JSR [CHECK.RESULT]                 ;CHECK OS FOR CORRECT DATA
U 14CD, 894C,F4 ;17082          JMP [ERRORLOOP.T27.5]              ;LOOP ON ERROR IF ENABLED
                ;17083
U 14CE, 094D,A4 ;17084          JMP [END.T27]                      ;DONE
                ;17085
                ;17086  ERRORLOOP.T27.5:
U 14CF, B652,15 ;17087          MOV LS[#200] TO WR[0]              ;200 TO WR 0
U 14D0, 2100,15 ;17088          DEC WR[0]                          ;1FF IN WR 0
U 14D1, 3E20,15 ;17089          MOV WR[0] TO LS[PC]                ;OLD PC FOR ERROR 5 LOOP
U 14D2, 1B21,75 ;17090          MEM.REQ[READ.V.RCHK.IFILL] ADRS[PC] DT[LONG]
                ;17091                                             ;FILL IB WITH OLD DATA AND SET PC=3
U 14D3, 0A1B,4C ;17092          JSR [NOP.DELAY]                    ;EXECUTE 5 CYCLE DELAY TO ALLOW IB FILL
                ;17093                                             ; TO FINISH
U 14D4, 8401,4E ;17094          MOV IB.DATA TO OS                  ;DO PREFETCH WITH PAGE BOUNDARY
U 14D5, DB00,15 ;17095          NOP                                ;SKIP IF IB VALID WILL SKIP THIS
U 14D6, B652,15 ;17096          MOV LS[#200] TO WR[0]              ;200 IN WR 0
```

```
U 14D7, C042,15 ;17097          ADD LS[#2] TO WR[0]             ;202 IN WR
U 14D8, 3E20,15 ;17098          MOV WR[0] TO LS[PC]             ;SET UP MACRO PC FOR MOV IB.DATA
U 14D9, 894C,64 ;17099          JMP [LOOP.T27.5]                ;LOOP ON ERROR
                ;17100
                ;17101  END.T27:
```

```
;17102  .PAGE   "  TEST 28 Abort IB Fill on Uncorrectable Error (M8391  MCT Module)"
;17103  ;++
;17104  ;
;17105  ; Test Description:
;17106  ;
;17107  ;       This test checks that an uncorrectable error in a memory location that
;17108  ;       is accessed for a IB fill operation will cause the IB Fill to be abort-
;17109  ;       ed.  Main memory location 4 is written with data of all 0's with an
;17110  ;       uncorrectable error coded in the ckbts.  The error is written using the
;17111  ;       MAINT.ECC.DATA routine.  Location 0 is written with data of all 1's
;17112  ;       and correct ckbts.  The test will issue a MEM.REQ with MF=READ.V.RCHK
;17113  ;       .IFILL and DT=LONGWORD.  The address used is address 0 (the MME bit in
;17114  ;       CSR 1 was cleared previously so that a physical address access occurs).
;17115  ;       The IB is checked for all 1's. Then a MEM.REQ with MF=READ.V.RCHK.IFILL
;17116  ;       is issued again with address 4 as the destination.  Since address 4
;17117  ;       contains an uncorrectable error, the hardware should prevent the IB
;17118  ;       from loading by disabling LOAD IB H.  STALL MBSY L and ERROR L together
;17119  ;       perform this function.  CSR 1 will be checked to assure that only the
;17120  ;       RDS bit is set.  Then CSR 0 is checked for correct CKBTS on the double
;17121  ;       error (this is mainly a micro-code check).
;17122  ;
;17123  ; Assumptions:
;17124  ;
;17125  ;       It is assumed that all previous tests have run successfully.
;17126  ;
;17127  ; Test Steps:
;17128  ;
;17129  ;       1)  Set up error mask, error number, and module number in LS (for
;17130  ;           error printouts) and clear previous error number in LS.  Set up
;17131  ;           error mask to check low 8 bits.
;17132  ;       2)  Load memory location 0 with all 1's.  Then load CSR 1 with ckbts
;17133  ;           of 1111101(B) (these must be inverted before writing in CSR 1) to
;17134  ;           produce a double error condition for data of all 0's.  Clear other
;17135  ;           bits of CSR.
;17136  ;       3)  Load a temporary LS location with a 4 and also set bit 0 (this is
;17137  ;           used to branch into the correct flow to write an error in memory).
;17138  ;       4)  Issue a MEM.REQ with MF=MAINT.ECC.DATA and DT=LONGWORD.  Use the
;17139  ;           temporary LS location as the address.  Then issue a WRITE.MEM LS
;17140  ;           with data of 0's.
;17141  ;       5)  Clear LS 10 (used as address for MOV IB.DATA TO OS) and issue a
;17142  ;           MEM.REQ with MF=READ.V.RCHK.IFILL.  Use a location in LS containing
;17143  ;           0's as the address.  Issue a MOV IB.DATA TO OS and check OS for all
;17144  ;           1's.
;17145  ;       6)  Repeat step 5 using an LS location containing a 4 as the address.
;17146  ;           OS should still contain 1's.
;17147  ;       7)  Change error mask to check CSR bits (14-23 and 25-31) and check
;17148  ;           that only bit 31 (RDS) is set.
;17149  ;       8)  Change error mask to check bits 6-0 only and read CSR 0.  Check for
;17150  ;           correct CKBTs.
;17151  ;
;17152  ; Errors:
;17153  ;
;17154  ;
;17155  ;       Error 1 - IB FILL failed.  Expected and received data is contents of OS
;17156  ;                 which was loaded from IB reg (8 bits).
```

```
                    ;17157  ;       Error 2 - LOAD IB not inhibited by uncorrectable error in memory.
                    ;17158  ;       Error 3 - RDS clear or other bits set with an uncorrectable error in
                    ;17159  ;                 memory.  Expected and received data is CSR 1 bits 14-23 and
                    ;17160  ;                 bits 25-31.
                    ;17161  ;       Error 4 - CKBTs in CSR 0 incorrect for double bit error that occurred.
                    ;17162  ;                 Expected and received data is CSR 0 bits 0 - 6.
                    ;17163  ;
                    ;17164  ; Debug:
                    ;17165  ;
                    ;17166  ;       Two methods of debug will be explained.  If the MCT module is in a
                    ;17167  ;       test station then the MCT can be single stepped making debug easier
                    ;17168  ;       in some cases.  Otherwise only the CPU can be single stepped and this
                    ;17169  ;       method will be explained also.  When single stepping the memory con-
                    ;17170  ;       troller will help in debug, this section will explicitly suggest it.
                    ;17171  ;
                    ;17172  ;       Error 1 - Run previous tests again.
                    ;17173  ;
                    ;17174  ;       Error 2 - This error indicates a problem with the LOAD IB logic on the
                    ;17175  ;       MCT module.  While looping on error, check the inputs I LD IB L and
                    ;17176  ;       ENABLE ERROR H to the negated input 'AND' gate that produces LOAD IB H.
                    ;17177  ;       ENABLE ERROR H should be high whenever I LD IB L is low, preventing
                    ;17178  ;       LOAD IB H from asserting.  If incorrect, trace the signals back to
                    ;17179  ;       ERROR L and STALL MBSY L.  Both of these should be low when I LD IB L
                    ;17180  ;       is low.
                    ;17181  ;
                    ;17182  ;       Error 3 - This error indicates a problem with previously tested logic.
                    ;17183  ;       Run previous tests again.
                    ;17184  ;
                    ;17185  ;       Error 4 - This error most likely indicates a micro code failure.  Check
                    ;17186  ;       that CSR CB/SYN CLK is occurring in the cycle READ.V.RCHK.IFILL D.E.
                    ;17187  ;
                    ;17188  ;--
                    ;17189
                    ;17190  T.28:
U 14DA, B65E,15     ;17191          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;17192                                          ; STATUS WORD
U 14DB, 3E80,15     ;17193          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;17194                                          ; BIT 15 INDICATES START OF TEST TO
                    ;17195                                          ; CONSOLE
U 14DC, 10E0,15     ;17196          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;17197  WAIT.T28.0:
U 14DD, 894D,D4     ;17198          JMP [WAIT.T28.0]                        ;LOOP HERE WAITING FOR CONSOLE TO
                    ;17199                                          ; RESPOND
                    ;17200
U 14DE, 0A1A,9C     ;17201          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                    ;17202                                          ; CLEAR MCT CSR 1
U 14DF, B62D,95     ;17203          MOV LS[#FFFFFF00] TO WR[3]      ;LOW BYTE OF ZEROS IN WR
U 14E0, 3E8B,95     ;17204          MOV WR[3] TO LS[ERROR.MASK]     ;CHECK LOW BYTE ONLY FOR ERRORS
U 14E1, B62F,15     ;17205          MOV LS[CSR1.MASK] TO WR[2]      ;ERROR MASK FOR ERROR 3.  DO NOT CHECK
                    ;17206                                          ; BITS 0-13 OR BIT 24
                    ;17207
U 14E2, 999C,75     ;17208          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                    ;17209                                          ;SET UP TO WRITE ONES IN MEMORY AT
                    ;17210                                          ; PHYSICAL ADDRESS 0
U 14E3, 329E,15      17211          WRITE.MEM LS[ONES]              ;WRITE ALL ONES IN MAIN MEMORY 0
```

```
U 14E4, 370E,15 ;17212          MOV LS[CKBTS.1111101] TO WR[0]  ;GET DOUBLE ERROR CKBTS IN WR 0
U 14E5, A0C0,15 ;17213          COM WR[0]                       ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 14E6, 452C,15 ;17214          BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE
U 14E7, 8A17,FC ;17215          JSR [WRITE.CSR1]                ;PUT DOUBLE ERROR CKBTS IN CSR 1, CLEAR
                ;17216                                          ; ALL OTHER BITS IN CSR
U 14E8, 3644,15 ;17217          MOV LS[#4] TO WR[0]             ;PUT A 4 IN WR 0
U 14E9, C740,15 ;17218          BIS LS[BIT0] TO WR[0]           ;ALSO SET LVA 00 FOR BRANCH
U 14EA, BE12,15 ;17219          MOV WR[0] TO LS[T9]             ;STORE ADDRESS IN LS 9
U 14EB, 9D12,F5 ;17220          MEM.REQ[MAINT.ECC.DATA] ADRS[T9] DT[LONG]
                ;17221                                          ;SET UP TO USE DIAGNOSTIC ROUTINE
U 14EC, B29C,15 ;17222          WRITE.MEM LS[ZERO]              ;WRITE 0'S AT LOCATION 4 WITH A DOUBLE
                ;17223                                          ; BIT ERROR ENCODED IN CKBTS
                ;17224  LOOP.T28.1:
U 14ED, 369E,95 ;17225          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 14EE, E520,15 ;17226          CLR LS[PC]                      ;BYTE IN IB REG TO LOAD IN OS
U 14EF, 6512,15 ;17227          CLR LS[T9]                      ;ADDRESS IN MEMORY TO ACCESS
U 14F0, 9B13,75 ;17228          MEM.REQ[READ.V.RCHK.IFILL] ADRS[T9] DT[LONG]
                ;17229                                          ;SET UP TO LOAD IB FROM MAIN MEMORY
                ;17230                                          ; ADDRESS 0 (CONTAINS ONES)
U 14F1, 8401,4E ;17231          MOV IB.DATA TO OS               ;PUT IB REG IN OS
U 14F2, DB00,15 ;17232          NOP                             ;MOV IB.DATA DOES A SKIP IF IB VALID
U 14F3, B6F8,15 ;17233          MOV LS[OS] TO WR[0]             ;GET RESULT INTO WR 0 FOR CHECKING
U 14F4, 0869,3C ;17234          JSR [CHECK.RESULT]              ;CHECK FOR ONES
U 14F5, 894E,D4 ;17235          JMP [LOOP.T28.1]                ;ERROR LOOP IF ENABLED
                ;17236
                ;17237  LOOP.T28.3.4:
U 14F6, 3642,15 ;17238          MOV LS[#2] TO WR[0]             ;PUT A 2 IN WR
U 14F7, BE82,15 ;17239          MOV WR[0] TO LS[ERROR.NUMBER]   ;ERROR 2
U 14F8, 3E88,95 ;17240          MOV WR[3] TO LS[ERROR.MASK]     ;CHECK LOW 8 BITS ONLY (SET UP IN CASE
                ;17241                                          ; WE ARE LOOPING ON ERROR 3 OR 4)
                ;17242  LOOP.T28.2:
U 14F9, 369E,95 ;17243          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 14FA, E520,15 ;17244          CLR LS[PC]                      ;BYTE IN IB REG TO LOAD IN OS
U 14FB, 3644,15 ;17245          MOV LS[#4] TO WR[0]             ;ADDRESS IN MEMORY TO ACCESS
U 14FC, BE12,15 ;17246          MOV WR[0] TO LS[T9]             ;PUT IN LS
U 14FD, 9B13,75 ;17247          MEM.REQ[READ.V.RCHK.IFILL] ADRS[T9] DT[LONG]
                ;17248                                          ;SET UP TO LOAD IB FROM MAIN MEMORY
                ;17249                                          ; ADDRESS 4 (CONTAINS 0'S WITH A
                ;17250                                          ; RDS ERROR)
U 14FE, 8401,4E ;17251          MOV IB.DATA TO OS               ;PUT IB REG IN OS
U 14FF, DB00,15 ;17252          NOP                             ;IF IFILL IS NOT ABORTED, SKIP ON IB
                ;17253                                          ; VALID WILL OCCUR
U 1500, B6F8,15 ;17254          MOV LS[OS] TO WR[0]             ;GET RESULT INTO WR 0 FOR CHECKING
U 1501, 0869,3C ;17255          JSR [CHECK.RESULT]              ;CHECK FOR ONES (LOAD IB ABORTED)
U 1502, 894F,94 ;17256          JMP [LOOP.T28.2]                ;ERROR LOOP IF ENABLED
                ;17257
U 1503, FF82,15 ;17258          INC LS[ERROR.NUMBER]            ;ERROR 3
U 1504, B67E,95 ;17259          MOV LS[RDS] TO WR[1]            ;EXPECTED DATA (RDS SET, OTHERS CLEAR)
U 1505, BE88,15 ;17260          MOV WR[2] TO LS[ERROR.MASK]     ;CHECK CSR 1 ERROR AND CONTROL BITS
                ;17261                                          ; (BITS 14-23 AND 25-31)
U 1506, 9D45,75 ;17262          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;17263                                          ;SET UP TO READ CSR 1
U 1507, 3022,15 ;17264          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 1508, 0869,3C ;17265          JSR [CHECK.RESULT]              ;CHECK FOR RDS SET, OTHERS CLEAR
U 1509, 894F,64 ;17266          JMP [LOOP.T28.3.4]              ;ERROR LOOP IF ENABLED
```

```
                ;17267
U 150A, FF82,15 ;17268          INC LS[ERROR.NUMBER]              ;ERROR 4
U 150B, A006,15 ;17269          MOV WR[3] TO WR[0]                ;FFFFFF00 IN WR 0
U 150C, 474E,15 ;17270          BIS LS[BIT7] TO WR[0]             ;SET BIT 7 IN WR 0
U 150D, 3E8A,15 ;17271          MOV WR[0] TO LS[ERROR.MASK]       ;ERROR MASK CHECKS BITS 6-0 ONLY
U 150E, B70E,95 ;17272          MOV LS[CKBTS.1111101] TO WR[1]    ;GET DOUBLE ERROR CKBTS IN WR 1
U 150F, A0C2,95 ;17273          COM WR[1]                         ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
                ;17274                                            ; (EXPECTED DATA)
U 1510, 9D9D,75 ;17275          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                ;17276                                            ;SET UP TO READ CSR 0
U 1511, 3022,15 ;17277          MOV MEM.DATA TO WR[0]             ;READ CSR 0 FOR CORRECT CKBTS
U 1512, 0869,3C ;17278          JSR [CHECK.RESULT]                ;CHECK FOR ERROR
U 1513, 894F,64 ;17279          JMP [LOOP.T28.3.4]                ;ERROR LOOP IF ENABLED
                ;17280
                ;17281  END.T28:
```

```
;17282  .PAGE   "  TEST 29 Clear CSR and Init On CINIT, UBS DCLO (M8391 MCT or M8394 WCS Module)"
;17283  ;++
;17284  ;
;17285  ; Test Description:
;17286  ;
;17287  ;       This test checks the logic used to clear the CSR 1 control bits (bits
;17288  ;       25-29) and cause a PWRFL memory initialization.  Two signals that are
;17289  ;       generated on the WCS module are used to cause this: CINIT L and UBS
;17290  ;       DCLO L.  When these are asserted they travel through 2 latches to pro-
;17291  ;       duce L INIT DLY H 2 memory cycles later.  This signal produces CLR CSR
;17292  ;       L and is an input to the POWER UP AND INIT PAL.  This PAL is respons-
;17293  ;       ible for asserting PWRFL L which will force the memory micro-code to
;17294  ;       the power up initilization routine when either ALLOW REF is asserted
;17295  ;       or the idle loop is entered.
;17296  ;         The routine writes CSR 1 bits 25-29 to all ones and executes a MEM
;17297  ;       .REQ with MF=WRITE.TB.  Then a WRITE.MEM is executed with data of al-
;17298  ;       ternating 1's and 0's.  Next a MEM.REQ is issued with MF=READ.TB.
;17299  ;       A CPU ATTN is issued to the console to issue CINIT and control is
;17300  ;       passed back to the CPU.  A delay of 5 NOPS is executed and then a MOV
;17301  ;       MEM.DATA TO WR[0] is performed.  The data received should be different
;17302  ;       than that sent since a power fail would have occurred before the MOV
;17303  ;       MEM.DATA was issued.  The power fail removes the data which was re-
;17304  ;       quested by the MEM.REQ[READ.TB] from the MC bus.  Finally CSR 1 is
;17305  ;       checked for 0's in bits 29-25.  CINIT is cleared before reading the
;17306  ;       CSR.  The test is repeated with UBS DCLO asserted.
;17307  ;
;17308  ; Assumptions:
;17309  ;
;17310  ;       It is assumed that all previous tests have run successfully.
;17311  ;
;17312  ; Test Steps:
;17313  ;
;17314  ;       1)  Set up error mask, error number, and module number in LS (for
;17315  ;           error printouts) and clear previous error number in LS.
;17316  ;       2)  Write the TB with a alternating 1's and 0's pattern at location 0.
;17317  ;       3)  Write CSR 1 with 1's in bits 25-29.  Then issue a MEM.REQ with MF=
;17318  ;           READ.TB.  Issue CPU ATTN to request the console to assert CINIT.
;17319  ;       4)  After control is passed back to the CPU, execute five NOPs as a
;17320  ;           delay to let the MC bus settle.  Then issue a MOV MEM.DATA TO WR[0]
;17321  ;           and check that the result is not alternating 1's and 0's.
;17322  ;       5)  Issue CPU ATTN to turn off CINIT, change error mask, and read CSR
;17323  ;           1.  Check that bits 29-25 are cleared.
;17324  ;       6)  Repeat steps 2 through 5 with UBS DCLO asserted.
;17325  ;
;17326  ; Errors:
;17327  ;
;17328  ;       Error 1 - MCT failed to perform PWRFL with CINIT asserted.
;17329  ;       Error 2 - CSR 1 bits 29-25 failed to clear with CINIT asserted.
;17330  ;       Error 3 - MCT failed to perform PWRFL with UBS DCLO asserted.
;17331  ;       Error 4 - CSR 1 bits 29-25 failed to clear with UBS DCLO asserted.
;17332  ;
;17333  ; Debug:
;17334  ;
;17335  ;       Two methods of debug will be explained.  If the MCT module is in a
;17336  ;       test station then the MCT can be single stepped making debug easier
```

```
                    ;17337  ;       in some cases.  Otherwise only the CPU can be single stepped and this
                    ;17338  ;       method will be explained also.  When single stepping the memory con-
                    ;17339  ;       troller will help in debug, this section will explicitly suggest it.
                    ;17340  ;
                    ;17341  ;       Error 1 - This error indicates a problem with the POWER UP AND INIT
                    ;17342  ;       PAL, its inputs, or the logic which produces INIT DLY H from CINIT L.
                    ;17343  ;       Single step the CPU to the instruction preceeding the MOV MEM.DATA
                    ;17344  ;       TO WR[0] and check the signal L INIT DLY H at the input to the POWER
                    ;17345  ;       UP AND INIT PAL for a high.  If this is incorrect, trace it back
                    ;17346  ;       through the latches and negated input 'OR' gate to CINIT.  If CINIT
                    ;17347  ;       is high at this point, the problem may be on the WCS module.
                    ;17348  ;         If L INIT DLY H is high at the POWER UP AND INIT PAL, check the
                    ;17349  ;       input ALLOW REF H for a high.  If this is OK, the output PWRFL L
                    ;17350  ;       should be low.  If not, the PAL is bad.
                    ;17351  ;         If PWRFL L is low, check that it gets to the logic which produces
                    ;17352  ;       STOP MEM H.  If this is OK, check it at the input to the logic which
                    ;17353  ;       produces MUX SEL L.  The logic after that point has been previously
                    ;17354  ;       tested.
                    ;17355  ;
                    ;17356  ;       Error 2 - If this is the first error, it indicates a problem with
                    ;17357  ;       the inverter which produces CLR CSR L or the latch that outputs CSR 1
                    ;17358  ;       bits 29-25.  Follow the procedure above and trace back CLR CSR L if
                    ;17359  ;       incorrect at this latch.  If CSR low is low, the latch itself may be
                    ;17360  ;       bad.
                    ;17361  ;
                    ;17362  ;       Error 3 - Single step as in error 1 and check the signal UBS DCLO L
                    ;17363  ;       for a low.  Trace this up to the output of the negated input 'OR' gate.
                    ;17364  ;       The logic after this point has been tested previously.
                    ;17365  ;
                    ;17366  ;       Error 4 - See error 2.
                    ;17367  ;
                    ;17368  ;--
                    ;17369
                    ;17370  T.29:
                    ;17371
U 1514, B65E,15     ;17372          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;17373                                          ; STATUS WORD
U 1515, 3E80,15     ;17374          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;17375                                          ; BIT 15 INDICATES START OF TEST TO
                    ;17376                                          ; CONSOLE
U 1516, 10E0,15     ;17377          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;17378  WAIT.T29.0:
U 1517, 0951,74     ;17379          JMP [WAIT.T29.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;17380                                          ; RESPOND
                    ;17381
U 1518, 0A1A,9C     ;17382          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                    ;17383                                          ; CLEAR MCT CSR 1
U 1519, C740,15     ;17384          BIS LS[WCS] TO WR[0]            ;ALSO SET BIT FOR WCS
U 151A, 3E8C,15     ;17385          MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                    ;17386                                          ; MCT OR WCS BOARD
                    ;17387
U 151B, B640,15     ;17388          MOV LS[#1] TO WR[0]             ;1 IN WR
U 151C, BEFC,15     ;17389          MOV WR[0] TO LS[SIZE]           ;SET UP SIZE REG FOR DATA TYPE OF WORD
                    ;17390
U 151D, 989C,F5     ;17391          MEM.REQ[WRITE.TB] ADRS[ZERO] DT[LONG]
```

```
                    ;17392                                  ;SET UP TO WRITE TB ADDRESS 0
U 151E, B29A,15     ;17393          WRITE.MEM LS[#55555555] ;WRITE ALTERNATING 1'S AND ZEROS IN TB
U 151F, B69E,15     ;17394          MOV LS[ONES] TO WR[0]   ;SET UP DATA FOR CSR IN WR 0
U 1520, 8A17,FC     ;17395          JSR [WRITE.CSR1]        ;CALL ROUTINE TO WRITE 1'S IN CSR 1
                    ;17396                                  ; BITS 25-29
                    ;17397
                    ;17398  LOOP.T29.1.2:
U 1521, B640,15     ;17399          MOV LS[#1] TO WR[0]     ;1 IN WR
U 1522, BE82,15     ;17400          MOV WR[0] TO LS[ERROR.NUMBER] ;SET UP ERROR NUMBER IN LS
U 1523, 3660,15     ;17401          MOV LS[INTERUPT.EN] TO WR[0] ;SET BIT 16 IN WR
U 1524, 3E80,15     ;17402          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP TO TURN OFF CINIT (AND ANY
                    ;17403                                  ; OTHER INTERUPTS THAT ARE SET)
U 1525, 10E0,15     ;17404          MISC [SET.CP.ATTN]      ;ISSUE CPU ATTN TO CONSOLE
                    ;17405  WAIT.T29.1:
U 1526, 8952,64     ;17406          JMP [WAIT.T29.1]        ;LOOP HERE WAITING FOR CONSOLE TO
                    ;17407                                  ; RESPOND
                    ;17408
U 1527, 9C9D,F5     ;17409          MEM.REQ[READ.TB] ADRS[ZERO] DT[LONG]
                    ;17410                                  ;SET UP TO READ TB ADDRESS 0
U 1528, 3660,15     ;17411          MOV LS[INTERUPT.EN] TO WR[0] ;SET BIT 16 IN WR 0
U 1529, 4778,15     ;17412          BIS LS[CINIT] TO WR[0]  ;SET UP TO ISSUE CINIT FROM CONSOLE
U 152A, C76E,15     ;17413          BIS LS[NA.EXP.REC] TO WR[0] ;DON'T PRINT DATA UNDER EXP AND REC
U 152B, 3E80,15     ;17414          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL AND STATUS WORD
U 152C, 10E0,15     ;17415          MISC [SET.CP.ATTN]      ;ISSUE CPU ATTN TO CONSOLE
                    ;17416  WAIT.T29.2:
U 152D, 0952,D4     ;17417          JMP [WAIT.T29.2]        ;LOOP HERE WAITING FOR CONSOLE TO
                    ;17418                                  ; RESPOND
U 152E, 0A1B,4C     ;17419          JSR [NOP.DELAY]         ;EXECUTE 5 CYCLE DELAY TO ALLOW MC BUS
                    ;17420                                  ; TO SETTLE
U 152F, 2F82,95     ;17421          CLR WR[1]               ;SET UP FALSE EXPECTED DATA
U 1530, 3022,15     ;17422          MOV MEM.DATA TO WR[0]   ;GET DATA AFTER MEM INIT
                    ;17423          CMP LS[#55555555] WITH WR[0], ;SEE IF RESULT IS DATA SENT TO TB
                    ;17424                                  ; THE CINIT SHOULD HAVE PREVENTED THIS
U 1531, CE9A,55     ;17425           DT(SIZE)&SET.ALU.CC    ; SET CONDITION CODES ACCORDING TO LOW
                    ;17426                                  ; WORD
U 1532, 0953,51     ;17427          JMP.IF[NEQ] TO [TEST.T29.2] ;JMP IF NO ERROR
U 1533, 0869,3C     ;17428          JSR [CHECK.RESULT]      ;ELSE PRINT ERROR REPORT
U 1534, 0952,14     ;17429          JMP [LOOP.T29.1.2]      ;ERROR LOOP IF ENABLED
                    ;17430  TEST.T29.2:
U 1535, 3660,15     ;17431          MOV LS[INTERUPT.EN] TO WR[0] ;SET BIT 16 IN WR
U 1536, 3E80,15     ;17432          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP TO TURN OFF CINIT (AND ANY
                    ;17433                                  ; OTHER INTERUPTS THAT ARE SET)
U 1537, 10E0,15     ;17434          MISC [SET.CP.ATTN]      ;ISSUE CPU ATTN TO CONSOLE
                    ;17435  WAIT.T29.3:
U 1538, 8953,84     ;17436          JMP [WAIT.T29.3]        ;LOOP HERE WAITING FOR CONSOLE TO
                    ;17437                                  ; RESPOND
                    ;17438
U 1539, FF82,15     ;17439          INC LS[ERROR.NUMBER]    ;ERROR 2
U 153A, DF2A,15     ;17440          MCOM LS[#FF000000] TO WR[0] ;PUT 00FFFFFF IN WR 0
U 153B, 477E,15     ;17441          BIS LS[BIT31] TO WR[0]  ;SET BIT 31
U 153C, C77C,15     ;17442          BIS LS[BIT30] TO WR[0]  ;SET BIT 30
U 153D, C770,15     ;17443          BIS LS[BIT24] TO WR[0]  ;SET BIT 24
U 153E, 3E8A,15     ;17444          MOV WR[0] TO LS[ERROR.MASK] ;CHECK BITS 29-24
                    ;17445
U 153F, 2F82,95     ;17446          CLR WR[1]               ;EXPECTED DATA
```

```
U 1540, 9D45,75 ;17447          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;17448                                  ;SET UP TO READ CSR
U 1541, 3022,15 ;17449          MOV MEM.DATA TO WR[0]   ;GET CSR 1 DATA
U 1542, 0869,3C ;17450          JSR [CHECK.RESULT]      ;SEE IF BITS 29-24 IS CLEAR
U 1543, 0952,14 ;17451          JMP [LOOP.T29.1.2]      ;ERROR LOOP IF ENABLED
                ;17452
U 1544, E58A,15 ;17453          CLR LS[ERROR.MASK]      ;CHECK ALL BITS
U 1545, 989C,F5 ;17454          MEM.REQ[WRITE.TB] ADRS[ZERO] DT[LONG]
                ;17455                                  ;SET UP TO WRITE TB ADDRESS 0
U 1546, B29A,15 ;17456          WRITE.MEM LS[#55555555] ;WRITE ALTERNATING 1'S AND ZEROS IN TB
U 1547, B69E,15 ;17457          MOV LS[ONES] TO WR[0]   ;SET UP DATA FOR CSR IN WR 0
U 1548, 8A17,FC ;17458          JSR [WRITE.CSR1]        ;CALL ROUTINE TO WRITE 1'S IN CSR 1
                ;17459                                  ; BITS 25-29
                ;17460
                ;17461  LOOP.T29.3.4:
U 1549, 36C0,15 ;17462          MOV LS[#3(H)] TO WR[0]  ;3 IN WR
U 154A, BE82,15 ;17463          MOV WR[0] TO LS[ERROR.NUMBER] ;ERROR 3
U 154B, 3660,15 ;17464          MOV LS[INTERUPT.EN] TO WR[0] ;SET BIT 16 IN WR
U 154C, 3E80,15 ;17465          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP TO TURN OFF CINIT (AND ANY
                ;17466                                  ; OTHER INTERUPTS THAT ARE SET)
U 154D, 10E0,15 ;17467          MISC [SET.CP.ATTN]      ;ISSUE CPU ATTN TO CONSOLE
                ;17468  WAIT.T29.4:
U 154E, 0954,E4 ;17469          JMP [WAIT.T29.4]        ;LOOP HERE WAITING FOR CONSOLE TO
                ;17470                                  ; RESPOND
                ;17471
U 154F, 9C9D,F5 ;17472          MEM.REQ[READ.TB] ADRS[ZERO] DT[LONG]
                ;17473                                  ;SET UP TO READ TB ADDRESS 0
U 1550, 3660,15 ;17474          MOV LS[INTERUPT.EN] TO WR[0] ;SET BIT 16 IN WR 0
U 1551, C77A,15 ;17475          BIS LS[UBS.DCLO] TO WR[0] ;SET UP TO ISSUE DC LO FROM CONSOLE
U 1552, C76E,15 ;17476          BIS LS[NA.EXP.REC] TO WR[0] ;DON'T PRINT DATA UNDER EXP AND REC
U 1553, 3E80,15 ;17477          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL AND STATUS WORD
U 1554, 10E0,15 ;17478          MISC [SET.CP.ATTN]      ;ISSUE CPU ATTN TO CONSOLE
                ;17479  WAIT.T29.5:
U 1555, 0955,54 ;17480          JMP [WAIT.T29.5]        ;LOOP HERE WAITING FOR CONSOLE TO
                ;17481                                  ; RESPOND
U 1556, 0A1B,4C ;17482          JSR [NOP.DELAY]         ;EXECUTE 5 CYCLE DELAY TO ALLOW MC BUS
                ;17483                                  ; TO SETTLE
U 1557, 2F82,95 ;17484          CLR WR[1]               ;SET UP FALSE EXPECTED DATA
U 1558, 3022,15 ;17485          MOV MEM.DATA TO WR[0]   ;GET DATA AFTER MEM INIT
                ;17486          CMP LS[#55555555] WITH WR[0], ;SEE IF RESULT IS DATA SENT TO TB
                ;17487                                  ; THE DC LO SHOULD HAVE PREVENTED THIS
U 1559, CE9A,35 ;17488           DT(LONG)&SET.ALU.CC    ; SET CONDITION CODES
U 155A, 8955,D1 ;17489          JMP.IF[NEQ] TO [TEST.T29.4] ;JMP IF NO ERROR
U 155B, 0869,3C ;17490          JSR [CHECK.RESULT]      ;ELSE PRINT ERROR REPORT
U 155C, 8954,94 ;17491          JMP [LOOP.T29.3.4]      ;ERROR LOOP IF ENABLED
                ;17492  TEST.T29.4:
U 155D, 3660,15 ;17493          MOV LS[INTERUPT.EN] TO WR[0] ;SET BIT 16 IN WR
U 155E, 3E80,15 ;17494          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP TO TURN OFF DC LO (AND ANY
                ;17495                                  ; OTHER INTERUPTS THAT ARE SET)
U 155F, 10E0,15 ;17496          MISC [SET.CP.ATTN]      ;ISSUE CPU ATTN TO CONSOLE
                ;17497  WAIT.T29.6:
U 1560, 0956,04 ;17498          JMP [WAIT.T29.6]        ;LOOP HERE WAITING FOR CONSOLE TO
                ;17499                                  ; RESPOND
                ;17500
U 1561, FF82,15 ;17501          INC LS[ERROR.NUMBER]    ;ERROR 4
```

```
U 1562, DF2A,15 ;17502          MCOM LS[#FF000000] TO WR[0]     ;PUT 00FFFFFF IN WR 0
U 1563, 477E,15 ;17503          BIS LS[BIT31] TO WR[0]          ;SET BIT 31
U 1564, C77C,15 ;17504          BIS LS[BIT30] TO WR[0]          ;SET BIT 30
U 1565, C770,15 ;17505          BIS LS[BIT24] TO WR[0]          ;SET BIT 24
U 1566, 3E8A,15 ;17506          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK BITS 29-24
                ;17507
U 1567, 2F82,95 ;17508          CLR WR[1]                       ;EXPECTED DATA
U 1568, 9D45,75 ;17509          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;17510                                          ;SET UP TO READ CSR
U 1569, 3022,15 ;17511          MOV MEM.DATA TO WR[0]           ;GET CSR 1 DATA
U 156A, 0869,3C ;17512          JSR [CHECK.RESULT]              ;SEE IF BITS 29-24 IS CLEAR
U 156B, 8954,94 ;17513          JMP [LOOP.T29.3.4]              ;ERROR LOOP IF ENABLED
                ;17514
                ;17515  END.T29:
```

```
;17516  .page   "Micro-flow Verification Tests"
;17517  .toc    "  TEST 2A Read Array Micro-flow Test  (M8391 MCT Module)"
;17518  ;++
;17519  ;
;17520  ; Test Description:
;17521  ;
;17522  ;       This test is designed to check for faults in the control store PROMs
;17523  ;       by verifying all paths of the micro-code.  This test checks the READ
;17524  ;       ARRAY flows for a delayed CPU DR response and for a 2 cycle read.
;17525  ;       Neither of these flows has been used before.
;17526  ;         The delayed CPU DR flow is checked by executing a READ P with NOPs
;17527  ;       after the MEM.REQ to delay the CPU DATA REQ.  This forces the micro-
;17528  ;       code to take the CPU DR NOT*ERROR SUM NOT path at the CPU RD V CONT C7
;17529  ;       cycle.  It also takes the CPU DR NOT*ERR NOT branch in the following
;17530  ;       cycle.
;17531  ;         The 2 cycle read flow is checked by doing a longword read at physical
;17532  ;       address 2.  This should access the last 2 bytes of longword 0, and the
;17533  ;       first 2 bytes of address 4.
;17534  ;
;17535  ; Assumptions:
;17536  ;
;17537  ;       It is assumed that all previous tests have run successfully, and
;17538  ;       that all hardware (other than MCT control store PROMs) is operating
;17539  ;       correctly.
;17540  ;
;17541  ; Test Steps:
;17542  ;
;17543  ;       1) Set up error mask, error number, and module number in LS (for
;17544  ;          error printouts) and clear previous error number in LS.  Also
;17545  ;          clear CSR 1 so that no diagnostic modes are enabled.
;17546  ;       2) Write data of FFFF0000 in address 0 and 0000FFFF in address 4.
;17547  ;       3) Execute a MEM.REQ[READ.P] to address 0 with DT=LONGWORD.  Execute
;17548  ;          NOPs to delay CPU DR, then issue MOV MEM.DATA TO WR[0] and check
;17549  ;          the data.  Repeat with address 4.
;17550  ;       4) Execute MEM.REQ[READ.P] at address 2 with DT=LONGWORD.  Check result
;17551  ;          for all 1's
;17552  ;
;17553  ; Errors:
;17554  ;
;17555  ;       NOTE:  Expected and received data is ARRAY data.
;17556  ;
;17557  ;       Error 1 - READ ARRAY flow to address 0 with CPU DR delayed failed (data
;17558  ;                 of FFFF0000).
;17559  ;       Error 2 - READ ARRAY flow to address 4 with CPU DR delayed failed (data
;17560  ;                 of 0000FFFF).
;17561  ;       Error 3 - TWO CYCLE READ ARRAY flow to address 2 failed.
;17562  ;
;17563  ; Debug:
;17564  ;
;17565  ;       Errors 1, 2, and 3 - The hardware associated with these reads has been
;17566  ;       tested previously.  Suspect the MCT control store PROMs if an error
;17567  ;       occurs.  The best way to debug an error is to single step the MCT
;17568  ;       and check that the proper paths are taken.  If this is OK, the expected
;17569  ;       result at each step will have to be verified.
;17570  ;
```

```
                ;17571  ;--
                ;17572
                ;17573  T.2A:
U 156C, B65E,15 ;17574          MOV LS[BEGIN.TEST] TO WR[0]      ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;17575                                           ; STATUS WORD
U 156D, 3E80,15 ;17576          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;17577                                           ; BIT 15 INDICATES START OF TEST TO
                ;17578                                           ; CONSOLE
U 156E, 10E0,15 ;17579          MISC [SET.CP.ATTN]               ;ISSUE CPU ATTN TO CONSOLE
                ;17580  WAIT.T2A.0:
U 156F, 0956,F4 ;17581          JMP [WAIT.T2A.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                ;17582                                           ; RESPOND
                ;17583
U 1570, 0A1A,9C ;17584          JSR [SETUP]                      ;SET UP MASKS, MODULE CODE ETC. AND
                ;17585                                           ; CLEAR MCT CSR 1
                ;17586
U 1571, 5F28,15 ;17587          MCOM LS[#FFFF] TO WR[0]          ;PUT FFFF0000 IN WR
U 1572, BE12,15 ;17588          MOV WR[0] TO LS[T9]              ;STORE DATA IN LS
U 1573, 999C,75 ;17589          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;17590                                           ;SET UP TO WRITE ADDRESS 0
U 1574, B212,15 ;17591          WRITE.MEM LS[T9]                 ;WRITE FFFF0000 IN MEMORY
U 1575, 9944,75 ;17592          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                ;17593                                           ;SET UP TO WRITE ADDRESS 4
U 1576, B228,15 ;17594          WRITE.MEM LS[#FFFF]              ;WRITE 0000FFFF IN MEMORY
                ;17595  LOOP.T2A.1:
U 1577, 189D,F5 ;17596          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;17597                                           ;SET UP TO READ ADDRESS 0
U 1578, 0A1B,4C ;17598          JSR [NOP.DELAY]                  ;EXECUTE 5 CYCLE DELAY TO DELAY CPU DR
                ;17599                                           ; SO WILL TAKE NEW MICRO-CODE PATH
U 1579, DF28,95 ;17600          MCOM LS[#FFFF] TO WR[1]          ;EXPECTED DATA (FFFF0000)
U 157A, 3022,15 ;17601          MOV MEM.DATA TO WR[0]            ;GET RESULT FROM MEMORY
U 157B, 0869,3C ;17602          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 157C, 0957,74 ;17603          JMP [LOOP.T2A.1]                 ;ERROR LOOP IF ENABLED
                ;17604
U 157D, FF82,15 ;17605          INC LS[ERROR.NUMBER]             ;ERROR 2
                ;17606  LOOP.T2A.2:
U 157E, 1845,F5 ;17607          MEM.REQ[READ.P] ADRS[#4] DT[LONG]
                ;17608                                           ;SET UP TO READ ADDRESS 4
U 157F, 0A1B,4C ;17609          JSR [NOP.DELAY]                  ;EXECUTE 5 CYCLE DELAY TO DELAY CPU DR
                ;17610                                           ; SO WILL TAKE NEW MICRO-CODE PATH
U 1580, B628,95 ;17611          MOV LS[#FFFF] TO WR[1]           ;EXPECTED DATA
U 1581, 3022,15 ;17612          MOV MEM.DATA TO WR[0]            ;GET RESULT FROM MEMORY
U 1582, 0869,3C ;17613          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 1583, 0957,E4 ;17614          JMP [LOOP.T2A.2]                 ;ERROR LOOP IF ENABLED
                ;17615
U 1584, FF82,15 ;17616          INC LS[ERROR.NUMBER]             ;ERROR 3
                ;17617  LOOP.T2A.3:
U 1585, 1843,F5 ;17618          MEM.REQ[READ.P] ADRS[#2] DT[LONG]
                ;17619                                           ;SET UP FOR 2 CYCLE READ STARTING AT
                ;17620                                           ; ADDRESS 2
U 1586, 369E,95 ;17621          MOV LS[ONES] TO WR[1]            ;EXPECTED DATA
U 1587, 3022,15 ;17622          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 1588, 0869,3C ;17623          JSR [CHECK.RESULT]               ;CHECK FOR ERROR
U 1589, 8958,54 ;17624          JMP [LOOP.T2A.3]                 ;ERROR LOOP IF ENABLED
                ;17625
```

```
;17626  END.T2A:
```

```
;17627  .page   "  TEST 2B Read Array Flow with Single/Double Bit Error (M8391 MCT Module)"
;17628  ;++
;17629  ;
;17630  ; Test Description:
;17631  ;
;17632  ;       This test is designed to check for faults in the control store PROMs
;17633  ;       by verifying all paths of the micro-code.  This test checks the path
;17634  ;       taken when single or double bit errors are detected in memory.  The
;17635  ;       test writes a single or double bit error in memory by using the MAINT
;17636  ;       .ECC.DATA routine.  CKBTS for all 0's are written with data of 10000(H)
;17637  ;       or 30000(H) to produce a single or double bit error respectively.  Then
;17638  ;       a read to that location is performed.  For the single bit error, the
;17639  ;       data should be corrected to all 0's and the resulting SYNDROME bits
;17640  ;       should be clocked into CSR 0.  CSR 1 should have the CRD bit set, and
;17641  ;       all other error bits cleared.  On the double error, the data should be
;17642  ;       unchanged, CSR 0 should get the CKBTs, and CSR 1 should have the RDS
;17643  ;       bit set and other error bits clear.  The test is performed twice, once
;17644  ;       with no delay between MEM.REQ[READ.P] and the MOV MEM.DATA WR[C], and
;17645  ;       once with NOPs between these two instructions.  This will check the
;17646  ;       different paths taken when CPU DR is present or delayed.  The two cycle
;17647  ;       read is also checked with single and double bit errors (but no delay
;17648  ;       since it uses the same path as a 1 cycle read as far as CPU DR is con-
;17649  ;       cerned).
;17650  ;
;17651  ; Assumptions:
;17652  ;
;17653  ;       It is assumed that all previous tests have run successfully, and
;17654  ;       that all hardware (other than MCT control store PROMs) is operating
;17655  ;       correctly.
;17656  ;
;17657  ; Test Steps:
;17658  ;
;17659  ;       1)  Set up error mask, error number, and module number in LS (for
;17660  ;           error printouts) and clear previous error number in LS.
;17661  ;       2)  Write all 1's at address 4.
;17662  ;       3)  Load CSR 1 with ckbts of 0111100(B) (these must be inverted before
;17663  ;           writing in CSR 1).  These are ckbts for data of all 0's.  Clear
;17664  ;           other bits of CSR 1.
;17665  ;       4)  Issue a MEM.REQ with MF=MAINT.ECC.DATA and DT=LONGWORD.  Use an
;17666  ;           address of 1 (LVA00 set) to write address 0 (LVA00 is used as a
;17667  ;           branch condition in the MAINT.ECC.DATA flow).Then issue a WRITE.MEM
;17668  ;           LS with data of a 10000(H) (1 in bit 0 of 3rd byte)
;17669  ;       5)  Read location 0 using the READ.P flow.  Check the data for all 0's
;17670  ;           (corrected data).
;17671  ;       6)  Change error mask and check CSR 1 for CRD set, others clear.
;17672  ;       7)  Change error mask and check CSR 0 for correct syndrome bits.
;17673  ;       8)  Repeat steps 5 through 7 with a delay of NOPs between the MEM.REQ
;17674  ;           and the MOV MEM.DATA to delay CPU DR.  This will take a different
;17675  ;           path in the microcode.
;17676  ;       9)  Read location 2 using a READ.P flow (two cycle read).  Check data
;17677  ;           for FFFF0000 (corrected data).  Then repeat steps 6 & 7.
;17678  ;       10) Repeat steps 3 through 8 above except write data of 30000(H) into
;17679  ;           address 0 (double error), check for no data correction, for RDS
;17680  ;           set, and for CKBTS in CSR 0 (instead of syndromes).
;17681  ;       11) Read location 2 using a READ.P flow (two cycle read).  Check data
```

```
;17682  ;           for 3(H) (no correction).  Then check CSR 1 for RDS set and CSR 0
;17683  ;           for CKBTS.
;17684  ;
;17685  ; Errors:
;17686  ;
;17687  ;       Error 1 - Single bit error not corrected correctly.  Expected and re-
;17688  ;                 ceived data is array contents.
;17689  ;       Error 2 - Single bit error failed to set CRD, or it set other bits in
;17690  ;                 CSR 1.  Expected and received data is CSR 1 bits 14-23 and
;17691  ;                 bits 25-31.
;17692  ;       Error 3 - Single bit error failed to set syndrome bits in CSR 0 cor-
;17693  ;                 rectly.  Expected and received data is CSR 0 bits 0-6.
;17694  ;       Error 4,5, or 6 - Same as error 1,2, or 3 except with delayed CPU DR.
;17695  ;       Error 7,8, and 9 -Same as error 1,2, or 3 except with 2 cycle read to
;17696  ;                 address 2.
;17697  ;       Error A - Memory data modified on double bit error.  Expected and re-
;17698  ;                 ceived data is array contents.
;17699  ;       Error B - Double bit error failed to set RDS or it set other bits in
;17700  ;                 CSR 1.  Expected and received data is CSR 1 bits 14-23 and
;17701  ;                 bits 25-31.
;17702  ;       Error C - Double bit error failed to set CKBTs in CSR 0 correctly.
;17703  ;                 Expected and received data is CSR 0 bits 0-6.
;17704  ;       Error D,E, or F - Same as error A,B, or C except with delayed CPU DR.
;17705  ;       Error 10,11, or 12 - Same as error A,B, or C except with 2 cycle read
;17706  ;                 to address 2.
;17707  ;
;17708  ; Debug:
;17709  ;
;17710  ;       Note:  The following errors most likely indicate a problem with the
;17711  ;       micro-code flow.  The hardware used by the micro-code has been pre-
;17712  ;       viously tested.  To trace a problem, check the sequence outlined
;17713  ;       below.  If the sequence is OK, suspect a bad bit in one of the control
;17714  ;       store PROMs and check each state for the correct enables.
;17715  ;
;17716  ;       Errors 1 through 3 - This indicates an error through the micro-code
;17717  ;       path that goes from CPU RD V CONT C7 to C8 to ERR C9, to SERR C10.
;17718  ;
;17719  ;       Errors 4 through 6 - This indicates an error through the micro-code
;17720  ;       path that goes from CPU RD V CONT C7 to C8A to ERR C9B to SERR C10.
;17721  ;
;17722  ;       Errors 7 through 9 - This indicates an error through the micro-code
;17723  ;       path that goes from ARY CYC C4 to 2 ARY CYC C5 to 2 ARY CYC RD C6 to
;17724  ;       2 ARY CYC RD C7  and then along the single error path.
;17725  ;
;17726  ;       Errors A through C - This indicates an error through the micro-code
;17727  ;       path that goes from CPU RD V CONT C7 to C8 to ERR C9 and then along
;17728  ;       the SERR NOT path.
;17729  ;
;17730  ;       Errors D through F - Same as errors 4 through 6 except takes the SERR
;17731  ;       NOT path instead of going to SERR C10.
;17732  ;
;17733  ;       Errors 10 through 12 - This indicates an error through the micro-code
;17734  ;       path that goes from ARY CYC C4 to 2 ARY CYC C5 to 2 ARY CYC RD C6 to
;17735  ;       2 ARY CYC RD C7  and then along the single error not path.
;17736  ;
```

```
                ;17737  ;--
                ;17738
                ;17739  T.2B:
U 158A, B65E,15 ;17740          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;17741                                          ; STATUS WORD
U 158B, 3E80,15 ;17742          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;17743                                          ; BIT 15 INDICATES START OF TEST TO
                ;17744                                          ; CONSOLE
U 158C, 10E0,15 ;17745          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;17746  WAIT.T2B.0:
U 158D, 0958,D4 ;17747          JMP [WAIT.T2B.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;17748                                          ; RESPOND
                ;17749
U 158E, 0A1A,9C ;17750          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                ;17751                                          ; CLEAR MCT CSR 1
U 158F, B713,95 ;17752          MOV LS[#FFFFFF80] TO WR[3]      ;SET BITS 7-31
U 1590, B62F,15 ;17753          MOV LS[CSR1.MASK] TO WR[2]      ;ERROR MASK FOR ERROR 3.  DO NOT CHECK
                ;17754                                          ; BITS 0-13 OR BIT 24
U 1591, B648,95 ;17755          MOV LS[#10] TO WR[1]            ;SET BIT 4
U 1592, 474A,95 ;17756          BIS LS[#20] TO WR[1]            ;AND BIT 5
U 1593, 474C,95 ;17757          BIS LS[#40] TO WR[1]            ;AND BIT 6 (1110000(B))
U 1594, A0C2,95 ;17758          COM WR[1]                       ;EXPECTED SYNDROMES
U 1595, BE10,95 ;17759          MOV WR[1] TO LS[T8]             ;SAVE IN TEMP LS LOCATION.
                ;17760
U 1596, 9944,75 ;17761          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                ;17762                                          ;SET UP TO WRITE AT ADDRESS 4
U 1597, 329E,15 ;17763          WRITE.MEM LS[ONES]              ;WRITE ALL 1'S IN MEMORY
U 1598, B700,15 ;17764          MOV LS[CKBTS.0111100] TO WR[0]  ;GET CKBTS FOR DATA OF 0 IN WR 0
U 1599, A0C0,15 ;17765          COM WR[0]                       ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 159A, 452C,15 ;17766          BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE
U 159B, 8A17,FC ;17767          JSR [WRITE.CSR1]                ;PUT CKBTS FOR DATA OF 0 IN CSR 1, CLEAR
                ;17768                                          ; ALL OTHER BITS IN CSR
U 159C, 1D40,F5 ;17769          MEM.REQ[MAINT.ECC.DATA] ADRS[#1] DT[LONG]
                ;17770                                          ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;17771                                          ; SET FOR BRANCH
U 159D, B260,15 ;17772          WRITE.MEM LS[#10000]            ;WRITE A 10000(H) AT LOCATION 0 WITH
                ;17773                                          ; CKBTS FOR DATA OF ALL 0'S
                ;17774  LOOP.T2B.2.3:
U 159E, B640,15 ;17775          MOV LS[#1] TO WR[0]             ;1 IN WR
U 159F, BE82,15 ;17776          MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
U 15A0, E58A,15 ;17777          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
                ;17778  LOOP.T2B.1:
U 15A1, 2F82,95 ;17779          CLR WR[1]                       ;EXPECTED DATA
U 15A2, 189D,F5 ;17780          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;17781                                          ;SET UP TO READ MEMORY ADDRESS 0
U 15A3, 3022,15 ;17782          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 15A4, 0869,3C ;17783          JSR [CHECK.RESULT]              ;CHECK FOR DATA OF 0
U 15A5, 895A,14 ;17784          JMP [LOOP.T2B.1]                ;ERROR LOOP IF ENABLED
                ;17785
U 15A6, 095E,9C ;17786          JSR [CHECK.CSRS.SINGLE]         ;CHECK CSR 1 AND CSR 0 (ERRORS 2 AND 3)
U 15A7, 8959,E4 ;17787          JMP [LOOP.T2B.2.3]              ;ERROR LOOP RETURN IF ENABLED
                ;17788
                ;17789  LOOP.T2B.5.6:
U 15A8, 3644,15 ;17790          MOV LS[#4] TO WR[0]             ;4 IN WR
U 15A9, BE82,15 ;17791          MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS (ERROR 4)
```

```
U 15AA, E58A,15 ;17792          CLR LS[ERROR.MASK]                ;CHECK ALL BITS
                ;17793  LOOP.T2B.4:
U 15AB, 2F82,95 ;17794          CLR WR[1]                         ;EXPECTED DATA
U 15AC, 189D,F5 ;17795          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;17796                                            ;SET UP TO READ MEMORY ADDRESS 0
U 15AD, 0A1B,4C ;17797          JSR [NOP.DELAY]                   ;EXECUTE 5 CYCLE DELAY TO DELAY SENDING
                ;17798                                            ; CPU DR
U 15AE, DB00,15 ;17799          NOP                               ;DELAY ANOTHER CYCLE
U 15AF, 3022,15 ;17800          MOV MEM.DATA TO WR[0]             ;GET RESULT
U 15B0, 0869,3C ;17801          JSR [CHECK.RESULT]                ;CHECK FOR DATA OF 0
U 15B1, 895A,B4 ;17802          JMP [LOOP.T2B.4]                  ;ERROR LOOP IF ENABLED
                ;17803
U 15B2, 095E,9C ;17804          JSR [CHECK.CSRS.SINGLE]           ;CHECK CSR 1 AND CSR 0 (ERRORS 5 AND 6)
U 15B3, 895A,84 ;17805          JMP [LOOP.T2B.5.6]                ;ERROR LOOP RETURN IF ENABLED
                ;17806
                ;17807  LOOP.T2B.8.9:
U 15B4, B646,15 ;17808          MOV LS[#8] TO WR[0]               ;8 IN WR
U 15B5, 2100,15 ;17809          DEC WR[0]                         ;ERROR 7
U 15B6, BE82,15 ;17810          MOV WR[0] TO LS[ERROR.NUMBER]     ;SET UP ERROR NUMBER IN LS
U 15B7, E58A,15 ;17811          CLR LS[ERROR.MASK]                ;CHECK ALL BITS
                ;17812  LOOP.T2B.7:
U 15B8, DF28,95 ;17813          MCOM LS[#FFFF] TO WR[1]           ;EXPECTED DATA (FFFF0000)
U 15B9, 1843,F5 ;17814          MEM.REQ[READ.P] ADRS[#2] DT[LONG]
                ;17815                                            ;SET UP TO READ MEMORY ADDRESS 2 (TWO
                ;17816                                            ; CYCLE READ)
U 15BA, 3022,15 ;17817          MOV MEM.DATA TO WR[0]             ;GET RESULT
U 15BB, 0869,3C ;17818          JSR [CHECK.RESULT]                ;CHECK FOR DATA OF FFFF0000
U 15BC, 095B,84 ;17819          JMP [LOOP.T2B.7]                  ;ERROR LOOP IF ENABLED
                ;17820
U 15BD, 095E,9C ;17821          JSR [CHECK.CSRS.SINGLE]           ;CHECK CSR 1 AND CSR 0 (ERRORS 8 AND 9)
U 15BE, 095B,44 ;17822          JMP [LOOP.T2B.8.9]                ;ERROR LOOP RETURN IF ENABLED
                ;17823
U 15BF, B700,15 ;17824          MOV LS[CKBTS.0111100] TO WR[0]    ;GET CKBTS FOR DATA OF 0 IN WR 0
U 15C0, A0C0,15 ;17825          COM WR[0]                         ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 15C1, 452C,15 ;17826          BIC LS[#FFFFFF00] TO WR[0]        ;CLEAR ALL BUT LOW BYTE
U 15C2, 8A17,FC ;17827          JSR [WRITE.CSR1]                  ;PUT CKBTS FOR DATA OF 0 IN CSR 1, CLEAR
                ;17828                                            ; ALL OTHER BITS IN CSR
                ;17829
U 15C3, 1D40,F5 ;17830          MEM.REQ[MAINT.ECC.DATA] ADRS[#1] DT[LONG]
                ;17831                                            ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;17832                                            ; SET FOR BRANCH
U 15C4, B334,15 ;17833          WRITE.MEM LS[#30000]              ;WRITE 30000(H) AT LOCATION 0 WITH CKBTS
                ;17834                                            ; FOR DATA OF ALL 0'S
                ;17835  LOOP.T2B.B.C:
U 15C5, B646,15 ;17836          MOV LS[#8] TO WR[0]               ;8 IN WR
U 15C6, C042,15 ;17837          ADD LS[#2] TO WR[0]               ;ERROR A
U 15C7, BE82,15 ;17838          MOV WR[0] TO LS[ERROR.NUMBER]     ;SET UP ERROR NUMBER IN LS
U 15C8, E58A,15 ;17839          CLR LS[ERROR.MASK]                ;CHECK ALL BITS
                ;17840  LOOP.T2B.A:
U 15C9, B734,95 ;17841          MOV LS[#30000] TO WR[1]           ;EXPECTED DATA
U 15CA, 189D,F5 ;17842          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;17843                                            ;SET UP TO READ MEMORY ADDRESS 0
U 15CB, 3022,15 ;17844          MOV MEM.DATA TO WR[0]             ;GET RESULT
U 15CC, 0869,3C ;17845          JSR [CHECK.RESULT]                ;CHECK FOR UNMODIFIED DATA
U 15CD, 095C,94 ;17846          JMP [LOOP.T2B.A]                  ;ERROR LOOP IF ENABLED
```

```
                    ;17847
U 15CE, 095F,8C     ;17848          JSR [CHECK.CSRS.DOUBLE]         ;CHECK CSR 1 AND CSR 0 (ERRORS B AND C)
U 15CF, 095C,54     ;17849          JMP [LOOP.T2B.B.C]              ;ERROR LOOP RETURN IF ENABLED
                    ;17850
                    ;17851  LOOP.T2B.E.F:
U 15D0, B646,15     ;17852          MOV LS[#8] TO WR[0]             ;8 IN WR
U 15D1, C044,15     ;17853          ADD LS[#4] TO WR[0]             ;C IN WR
U 15D2, 2040,15     ;17854          INC WR[0]                       ;ERROR D
U 15D3, BE82,15     ;17855          MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
U 15D4, E58A,15     ;17856          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
                    ;17857  LOOP.T2B.D:
U 15D5, B734,95     ;17858          MOV LS[#30000] TO WR[1]         ;EXPECTED DATA  (UNMODIFIED)
U 15D6, 189D,F5     ;17859          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                    ;17860                                          ;SET UP TO READ MEMORY ADDRESS 0
U 15D7, 0A1B,4C     ;17861          JSR [NOP.DELAY]                 ;EXECUTE 5 CYCLE DELAY TO DELAY SENDING
                    ;17862                                          ; CPU DR
U 15D8, DB00,15     ;17863          NOP                             ;DELAY ANOTHER CYCLE
U 15D9, 3022,15     ;17864          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 15DA, 0869,3C     ;17865          JSR [CHECK.RESULT]              ;CHECK FOR DATA OF 30000(H)
U 15DB, 895D,54     ;17866          JMP [LOOP.T2B.D]                ;ERROR LOOP IF ENABLED
                    ;17867
U 15DC, 095F,8C     ;17868          JSR [CHECK.CSRS.DOUBLE]         ;CHECK CSR 1 AND CSR 0 (ERRORS E AND F)
U 15DD, 895D,04     ;17869          JMP [LOOP.T2B.E.F]              ;ERROR LOOP RETURN IF ENABLED
                    ;17870
                    ;17871  LOOP.T2B.11.12:
U 15DE, 3648,15     ;17872          MOV LS[#10] TO WR[0]            ;ERROR 10
U 15DF, BE82,15     ;17873          MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
U 15E0, E58A,15     ;17874          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
                    ;17875  LOOP.T2B.10:
U 15E1, B6C0,95     ;17876          MOV LS[#3(H)] TO WR[1]          ;3(H) IN WR
U 15E2, 1843,F5     ;17877          MEM.REQ[READ.P] ADRS[#2] DT[LONG]
                    ;17878                                          ;SET UP TO READ MEMORY ADDRESS 2 (TWO
                    ;17879                                          ; CYCLE READ)
U 15E3, 3022,15     ;17880          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 15E4, 0869,3C     ;17881          JSR [CHECK.RESULT]              ;CHECK FOR UNMODIFIED DATA
U 15E5, 095E,14     ;17882          JMP [LOOP.T2B.10]               ;ERROR LOOP IF ENABLED
                    ;17883
U 15E6, 095F,8C     ;17884          JSR [CHECK.CSRS.DOUBLE]         ;CHECK CSR 1 AND CSR 0 (ERRORS 11 AND 12)
U 15E7, 095D,E4     ;17885          JMP [LOOP.T2B.11.12]            ;ERROR LOOP RETURN IF ENABLED
                    ;17886
U 15E8, 8960,84     ;17887          JMP [END.T2B]                   ;DONE
                    ;17888
                    ;17889
                    ;17890  CHECK.CSRS.SINGLE:
U 15E9, FF82,15     ;17891          INC LS[ERROR.NUMBER]            ;NEXT ERROR
U 15EA, BE88,15     ;17892          MOV WR[2] TO LS[ERROR.MASK]     ;ERROR MASK FOR CHECKING CSR 1
U 15EB, 367C,95     ;17893          MOV LS[CRD] TO WR[1]            ;EXPECTED DATA (CRD SET, OTHERS CLEAR)
U 15EC, 9D45,75     ;17894          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;17895                                          ;READ CSR1
U 15ED, 3022,15     ;17896          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 15EE, 0869,3C     ;17897          JSR [CHECK.RESULT]              ;CHECK FOR ONLY CRD SET
U 15EF, 5B00,14     ;17898          RETURN                          ;ERROR LOOP IF ENABLED
                    ;17899
U 15F0, FF82,15     ;17900          INC LS[ERROR.NUMBER]            ;NEXT ERROR
U 15F1, 3E8B,95     ;17901          MOV WR[3] TO LS[ERROR.MASK]     ;ERROR MASK FOR CHECKING CSR 0
```

```
U 15F2, 3610,95 ;17902          MOV LS[T8] TO WR[1]              ;EXPECTED SYNDROMES (1110000(B))
U 15F3, 9D9D,75 ;17903          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                ;17904                                           ;READ CSR 0
U 15F4, 3022,15 ;17905          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 15F5, 0869,3C ;17906          JSR [CHECK.RESULT]               ;CHECK FOR CORRECT SYNDROMES
U 15F6, 5B00,14 ;17907          RETURN                           ;ERROR LOOP IF ENABLED
U 15F7, DB00,16 ;17908          RETURN+1                         ;NO ERROR LOOP RETURN
                ;17909
                ;17910
                ;17911  CHECK.CSRS.DOUBLE:
U 15F8, FF82,15 ;17912          INC LS[ERROR.NUMBER]             ;NEXT ERROR
U 15F9, BE8B,15 ;17913          MOV WR[2] TO LS[ERROR.MASK]      ;ERROR MASK FOR CHECKING CSR 1
U 15FA, B67E,95 ;17914          MOV LS[RDS] TO WR[1]             ;EXPECTED DATA (RDS SET, OTHERS CLEAR)
U 15FB, 9D45,75 ;17915          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;17916                                           ;READ CSR1
U 15FC, 3022,15 ;17917          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 15FD, 0869,3C ;17918          JSR [CHECK.RESULT]               ;CHECK FOR ONLY RDS SET
U 15FE, 5B00,14 ;17919          RETURN                           ;ERROR LOOP RETURN IF ENABLED
                ;17920
U 15FF, FF82,15 ;17921          INC LS[ERROR.NUMBER]             ;NEXT ERROR
U 1600, 3E8B,95 ;17922          MOV WR[3] TO LS[ERROR.MASK]      ;ERROR MASK FOR CHECKING CSR 0
U 1601, 3700,95 ;17923          MOV LS[CKBTS.0111100] TO WR[1]   ;COMP OF EXPECTED DATA
U 1602, A0C2,95 ;17924          COM WR[1]                        ;EXPECTED DATA (CKBTS)
U 1603, 9D9D,75 ;17925          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                ;17926                                           ;READ CSR 0
U 1604, 3022,15 ;17927          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 1605, 0869,3C ;17928          JSR [CHECK.RESULT]               ;CHECK FOR CORRECT CKBTS
U 1606, 5B00,14 ;17929          RETURN                           ;ERROR LOOP RETURN IF ENABLED
U 1607, DB00,16 ;17930          RETURN+1                         ;NO ERROR LOOP RETURN
                ;17931
                ;17932  END.T2B:
```

```
;17933  .page   "  TEST 2C DIAG CHK and ECC DISABLE Mode Tests (M8391 MCT Module)"
;17934  ;++
;17935  ;
;17936  ; Test Description:
;17937  ;
;17938  ;       This test is designed to check for faults in the control store PROMs
;17939  ;       by verifying all paths of the micro-code.  This test checks the paths
;17940  ;       taken when DIAG CHK or ECC DIS or both are enabled via setting bits
;17941  ;       in CSR 1.
;17942  ;         When DIAG CHK mode is asserted, the CKBTs in bits 6-0 of CSR 1 are
;17943  ;       used in the ECC logic instead of the CKBTs read from memory.  This
;17944  ;       test will check that writing single error CKBTs in CSR1 will change
;17945  ;       the data read from a good memory location (due to the single bit error
;17946  ;       being "corrected").  It will also check that a double error will set
;17947  ;       the RDS bit without altering the data, and that CKBTS in CSR 1 that
;17948  ;       are correct for the data in memory will not alter anything or produce
;17949  ;       an error indication.
;17950  ;         When ECC DIS is asserted, the CKBTs from memory will be stored in CSR
;17951  ;       0.  If a single or double error occurs, the RDS bit in CSR 1 will be
;17952  ;       set.  No correction will occur on a single bit error.  This test will
;17953  ;       check the CKBTs in CSR 0, the CRD bit in CSR 1 and the data for cases
;17954  ;       of no error and single error.
;17955  ;         When both ECC DIS and DIAG CHK are set, the CKBTs in bits 6-0 of
;17956  ;       CSR 1 are used in the ECC logic as in DIAG CHK above.  The difference
;17957  ;       is that CSR 0 will be clocked with these CKBTs, and no correction will
;17958  ;       occur.  CRD will set in CSR 1 if a single error occurs.  This test will
;17959  ;       check CSR 0 for the CKBTs from CSR 1, and the CRD bit in CSR 1.
;17960  ;
;17961  ; Assumptions:
;17962  ;
;17963  ;       It is assumed that all previous tests have run successfully, and
;17964  ;       that all hardware (other than MCT control store PROMs) is operating
;17965  ;       correctly.
;17966  ;
;17967  ; Test Steps:
;17968  ;
;17969  ;       1) Set up error mask, error number, and module number in LS (for
;17970  ;          error printouts) and clear previous error number in LS.  Also
;17971  ;          clear CSR 1 so that no diagnostic modes are enabled.
;17972  ;       2) Write a 1 in memory location 0 and read to verify the data.
;17973  ;       3) Write the complement of CKBTs for data of all 0's in CSR 1
;17974  ;          (actual CKBTs of 0111100) and set the diag check mode bit.
;17975  ;       4) Read memory location 0 and check for corrected data of all 0's.
;17976  ;          This verifies the micro flow path which branches on diag check.
;17977  ;       5) Write CSR 1 with CKBT data for data of 1.  Also keep the diag
;17978  ;          check mode bit set.
;17979  ;       6) Read memory location 0 for data of 1.  Also check CSR 1 to verify
;17980  ;          that no error bits are set.  This verifies the micro flow path
;17981  ;          which branches on diag check and has no data error.
;17982  ;       7) Write data of 2 in memory location 0.  Read with CSR 1 containing
;17983  ;          CKBTs for data of a 1 with diag check mode set.  The data should
;17984  ;          come back uncorrected (double bit error).
;17985  ;       8) Read CSR 1 and check for RDS set.  This verifies the micro flow
;17986  ;          path that branches on diag check and branches again on SERR.
;17987  ;       9) Write data of all 0's in memory location 0.  Set the ECC DISABLE
```

```
;17988  ;          mode bit in CSR 1.  Read location 0 and check for data of 0's.
;17989  ;       10)Read CSR 0 for the complement of the CKBTs for data of 0's.  The
;17990  ;          actual CKBTs are 0111100.  This verifies the ECC DIS branch in
;17991  ;          the READ ARRAY micro flow.
;17992  ;       11)Read CSR 1 and check that all error bits are clear.
;17993  ;       12)Write data of 0's in memory with CKBTs for data of a 1 by using
;17994  ;          the ECC MAINT routine.
;17995  ;       13)Read location 0 with ECC DISABLE set in CSR 1.  Check that no
;17996  ;          correction occurs (read data of all 0's).
;17997  ;       14)Read CSR 0 for the complement of CKBTs for data of a 1.  The
;17998  ;          actual CKBTs should be 1100100(H).
;17999  ;       15)Read CSR 1 and verify that the CRD error bit is set.  This
;18000  ;          verifies the ERR branch in the ECC DISABLE flow.
;18001  ;       16)Write memory location 0 with data of a 1.  Write CSR 1 with
;18002  ;          CKBTS for data of 0 and set both ECC DIS and DIAG CHK bits.
;18003  ;       17)Read memory location 0 and check for uncorrected data (a 1).
;18004  ;       18)Check CSR 0 for the complement of CKBTs for data of 0 (actual
;18005  ;          CKBTs of 0111100(H)).  This verifies the branch on both ECC
;18006  ;          DIS and DIAG CHK set.
;18007  ;       19)Read CSR 1 and check for CRD set.  This verifies the ERR branch
;18008  ;          in the micro flow.
;18009  ;       20)Write CSR 1 with the complement of CKBTs for data of a 1 and set
;18010  ;          the ECC DIS and DIAG CHK bits.
;18011  ;       21)Read memory location 0, then read CSR 0 and check for CKBTS for
;18012  ;          data of 1 (actual CKBTs of 1100100(H)).
;18013  ;       22)Read CSR 1 and check that no error bits are set.  This verifies
;18014  ;          the ERR NOT branch in the micro flows.
;18015  ;
;18016  ; Errors:
;18017  ;
;18018  ;       NOTE:  Data under expected and received in error report is memory
;18019  ;       data when error mask is all 0's.  Data is CSR 1 contents when error
;18020  ;       mask is 07003FFF (H).  Data is CSR 0 contents when error mask is
;18021  ;       FFFFFF80 (H).
;18022  ;
;18023  ;       Error 1 - Write a 1, read a 1 from memory failed.
;18024  ;       Error 2 - Diag check mode flow failed to use CSR 1 CKBTs and execute
;18025  ;                 data correction.
;18026  ;       Error 3 - Diag check mode flow altered read data when CSR 1 contained
;18027  ;                 correct CKBTs.
;18028  ;       Error 4 - Diag check mode flow set error bits in CSR 1 when no error
;18029  ;                 should have occurred.
;18030  ;       Error 5 - Diag check mode flow altered read data on double error.
;18031  ;       Error 6 - Diag check mode flow failed to set RDS bit in CSR 1 on
;18032  ;                 double error, or set other error bits in CSR 1.
;18033  ;       Error 7 - ECC disable mode failed.  Altered data in memory.
;18034  ;       Error 8 - ECC disable mode failed to load memory CKBTs in CSR 0.
;18035  ;       Error 9 - ECC disable mode set an error indication in CSR 1 when no
;18036  ;                 error existed.
;18037  ;       Error A - ECC disable mode allowed error correction on single error.
;18038  ;       Error B - ECC disable mode failed to load memory CKBTs in CSR 0.
;18039  ;       Error C - ECC disable mode failed to set CRD error on single bit
;18040  ;                 error detection.
;18041  ;       Error D - ECC disable and Diag check mode altered memory data on single
;18042  ;                 bit error.
```

```
;18043  ;       Error E - ECC disable and Diag check mode failed to load CSR 0 from
;18044  ;                 CSR 1 correctly.
;18045  ;       Error F - ECC disable and Diag check mode failed to set error in CSR 1
;18046  ;                 on single bit error.
;18047  ;       Error 10- ECC disable and Diag check mode failed to load CSR 0 from
;18048  ;                 CSR 1 correctly (no single bit error in data).
;18049  ;       Error 11- ECC disable and Diag check mode set an error indication in
;18050  ;                 CSR 1 when no error existed.
;18051  ;
;18052  ; Debug:
;18053  ;
;18054  ;       Note:  The following errors most likely indicate a problem with the
;18055  ;       micro-code flow.  The hardware used by the micro-code has been pre-
;18056  ;       viously tested.  To trace a problem, check the sequence outlined
;18057  ;       below.  If the sequence is OK, suspect a bad bit in one of the control
;18058  ;       store PROMs and check each state for the correct enables.
;18059  ;
;18060  ;       Error 1 - This error indicates a problem with the normal memory
;18061  ;       write and read.  Run previous tests again if this fails.
;18062  ;
;18063  ;       Error 2 - This error indicates a problem with the DIAG CHECK branch
;18064  ;       or data error branches in the READ ARRAY micro flow.  The proper
;18065  ;       sequence should be from C6 to RD DIAG C7 to RD DIAG C7A to RD DIAG
;18066  ;       C8 to word that checks for an error to RD DIAG C10 to SERR C10.
;18067  ;
;18068  ;       Error 3 - This error follows the same flow as error 2 above except
;18069  ;       instead of going to RD DIAG C10, the flow goes to RD DIAG C11 and
;18070  ;       continues to a previously testes micro path.
;18071  ;
;18072  ;       Error 4 - Same as error 3.  CSR 1 data is checked in this error.
;18073  ;
;18074  ;       Error 5 - This error follows the same flow as error 2 above except
;18075  ;       instead of going to SERR C10, the single error not branch is taken.
;18076  ;
;18077  ;       Error 6 - Same as error 5 above except CSR 1 contents is checked.
;18078  ;
;18079  ;       Error 7 - This error indicates a problem with the ECC disable flow
;18080  ;       in the READ ARRAY micro code.  The correct path is from C6 to RD
;18081  ;       ECC DIS C7 to RD ECC DIS C8A to RD DIAG C11.
;18082  ;
;18083  ;       Error 8 - Same as error 7.
;18084  ;
;18085  ;       Error 9 - Same as error 7 (suspect the branch at RD ECC DIS C7.  It
;18086  ;       should take the no error path).
;18087  ;
;18088  ;       Error A - This error indicates a problem with the ECC disable flow
;18089  ;       in the READ ARRAY micro code.  The correct path is from C6 to RD
;18090  ;       ECC DIS C7 to RD ECC DIS C8 to RD ECC DIS C8A to RD DIAG C11.
;18091  ;
;18092  ;       Error B - Same as error A (suspect the ERR branch at RD ECC DIS C7.
;18093  ;       The path taken should be the ERR path).
;18094  ;
;18095  ;       Error C - Same as error A.
;18096  ;
;18097  ;       Error D - This error indicates a problem with the ECC disable and
```

```
                ;18098  ;       Diag check flow in the READ ARRAY micro code.  The correct path is
                ;18099  ;       from C6 to RD DIAG ECC DIS C7 to RD DIAG ECC DIS C8 to a word
                ;18100  ;       which branches on ERR to RD ECC DIS C8 to RD ECC DIS C8A to RD
                ;18101  ;       DIAG C11.
                ;18102  ;
                ;18103  ;       Error E - Same as error D (suspect the ECC DIS and DIAG CHK branch
                ;18104  ;       at C6).
                ;18105  ;
                ;18106  ;       Error F - Same as error D (suspect the ERR branch.  The flow should
                ;18107  ;       take the ERR path).
                ;18108  ;
                ;18109  ;       Error 10- This error indicates a problem with the ECC disable and
                ;18110  ;       Diag check flow in the READ ARRAY micro code.  The correct path is
                ;18111  ;       from C6 to RD DIAG ECC DIS C7 to RD DIAG ECC DIS C8 to a word
                ;18112  ;       which branches on ERR to RD ECC DIS C8A to RD DIAG C11.
                ;18113  ;
                ;18114  ;       Error 11- Same as error 10 (suspect the ERR branch.  The flow should
                ;18115  ;       take the no ERR path).
                ;18116  ;
                ;18117  ;--
                ;18118
                ;18119  T.2C:
U 1608, B65E,15 ;18120          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;18121                                          ; STATUS WORD
U 1609, 3E80,15 ;18122          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;18123                                          ; BIT 15 INDICATES START OF TEST TO
                ;18124                                          ; CONSOLE
U 160A, 10E0,15 ;18125          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;18126  WAIT.T2C.0:
U 160B, 8960,B4 ;18127          JMP [WAIT.T2C.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;18128                                          ; RESPOND
                ;18129
U 160C, 0A1A,9C ;18130          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                ;18131                                          ; CLEAR MCT CSR 1
U 160D, B713,95 ;18132          MOV LS[#FFFFFF80] TO WR[3]      ;SET BITS 7-31
U 160E, B62F,15 ;18133          MOV LS[CSR1.MASK] TO WR[2]      ;ERROR MASK FOR CSR 1
U 160F, C775,15 ;18134          BIS LS[DIAG.CHK] TO WR[2]       ;DISABLE CHECK ON DIAG.CHK
U 1610, C773,15 ;18135          BIS LS[ECC.DIS] TO WR[2]        ;DISABLE CHECK ON ECC DISABLE.  NOW WILL
                ;18136                                          ; NOT CHECK BITS 0-13, AND BITS 24-26
                ;18137
                ;18138  LOOP.T2C.1:
U 1611, 999C,75 ;18139          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;18140                                          ;SET UP TO WRITE MEMORY ADDRESS 0
U 1612, 3240,15 ;18141          WRITE.MEM LS[#1]                ;WRITE A 1 IN MEMORY
U 1613, 189D,F5 ;18142          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;18143                                          ;SET UP TO READ MEMORY ADDRESS 0
U 1614, 3022,15 ;18144          MOV MEM.DATA TO WR[0]           ;READ ADDRESS 0
U 1615, 3640,95 ;18145          MOV LS[#1] TO WR[1]             ;EXPECTED RESULT
U 1616, 0869,3C ;18146          JSR [CHECK.RESULT]              ;VERIFY MEMORY ADDRESS 0 CONTAINS A 1
U 1617, 0961,14 ;18147          JMP [LOOP.T2C.1]                ;ERROR LOOP IF ENABLED
                ;18148
U 1618, FF82,15 ;18149          INC LS[ERROR.NUMBER]            ;ERROR 2
U 1619, B700,15 ;18150          MOV LS[CKBTS.0111100] TO WR[0]  ;GET CKBTS FOR DATA OF 0 IN WR 0
U 161A, A0C0,15 ;18151          COM WR[0]                       ;COMP CKBTS TO CORRECT FOR HARDWARE
U 161B, 452C,15 ;18152          BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE
```

```
U 161C, 4774,15 ;18153          BIS LS[DIAG.CHK] TO WR[0]       ;SET DIAG CHK BIT
U 161D, 8A17,FC ;18154          JSR [WRITE.CSR1]                ;WRITE CKBTS AND DIAG CHK IN CSR1
                ;18155  LOOP.T2C.2:
U 161E, 189D,F5 ;18156          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;18157                                          ;SET UP TO READ MEMORY ADDRESS 0
U 161F, 3022,15 ;18158          MOV MEM.DATA TO WR[0]           ;READ ADDRESS 0
U 1620, 2F82,95 ;18159          CLR WR[1]                       ;EXPECTED RESULT (CORRECTED DATA)
U 1621, 0869,3C ;18160          JSR [CHECK.RESULT]              ;VERIFY MEMORY ADDRESS 0 CONTAINS A 0
U 1622, 0961,E4 ;18161          JMP [LOOP.T2C.2]                ;ERROR LOOP IF ENABLED
                ;18162
U 1623, 3704,15 ;18163          MOV LS[CKBTS.1100100] TO WR[0]  ;GET CKBTS FOR DATA OF 1 IN WR 0
U 1624, A0C0,15 ;18164          COM WR[0]                       ;COMP CKBTS TO CORRECT FOR HARDWARE
U 1625, 452C,15 ;18165          BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE
U 1626, 4774,15 ;18166          BIS LS[DIAG.CHK] TO WR[0]       ;SET DIAG CHK BIT
U 1627, 8A17,FC ;18167          JSR [WRITE.CSR1]                ;WRITE CKBTS AND DIAG CHK IN CSR1
                ;18168  LOOP.T2C.4:
U 1628, 36C0,15 ;18169          MOV LS[#3(H)] TO WR[0]          ;3 IN WR
U 1629, BE82,15 ;18170          MOV WR[0] TO LS[ERROR.NUMBER]   ;ERROR 3
U 162A, E58A,15 ;18171          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
                ;18172  LOOP.T2C.3:
U 162B, 189D,F5 ;18173          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;18174                                          ;SET UP TO READ MEMORY ADDRESS 0
U 162C, 3022,15 ;18175          MOV MEM.DATA TO WR[0]           ;READ ADDRESS 0
U 162D, 3640,95 ;18176          MOV LS[#1] TO WR[1]             ;EXPECTED RESULT (UNCORRECTED DATA)
U 162E, 0869,3C ;18177          JSR [CHECK.RESULT]              ;VERIFY MEMORY ADDRESS 0 CONTAINS A 1
U 162F, 0962,B4 ;18178          JMP [LOOP.T2C.3]                ;ERROR LOOP IF ENABLED
                ;18179
U 1630, FF82,15 ;18180          INC LS[ERROR.NUMBER]            ;ERROR 4
U 1631, BE8B,15 ;18181          MOV WR[2] TO LS[ERROR.MASK]     ;CSR 1 MASK CHECKS ERROR BITS
U 1632, 2F82,95 ;18182          CLR WR[1]                       ;EXPECTED DATA (ALL ERROR BITS CLEAR)
U 1633, 9D45,75 ;18183          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;18184                                          ;SET UP TO READ CSR 1
U 1634, 3022,15 ;18185          MOV MEM.DATA TO WR[0]           ;READ CSR1
U 1635, 0869,3C ;18186          JSR [CHECK.RESULT]              ;CHECK FOR CORRECT CSR1 CONTENTS
U 1636, 0962,84 ;18187          JMP [LOOP.T2C.4]                ;ERROR LOOP IF ENABLED
                ;18188
U 1637, FF82,15 ;18189          INC LS[ERROR.NUMBER]            ;ERROR 5
U 1638, 999C,75 ;18190          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;18191                                          ;SET UP TO WRITE MEMORY ADDRESS 0
U 1639, B242,15 ;18192          WRITE.MEM LS[#2]                ;WRITE A 2 IN MEMORY (BITS 0 AND 1
                ;18193                                          ; WILL BE INCORRECT THUS RDS ERROR)
U 163A, 3682,15 ;18194          MOV LS[ERROR.NUMBER] TO WR[0]   ;PUT IN WR
U 163B, 3E10,15 ;18195          MOV WR[0] TO LS[T8]             ;SAVE IN TEMP LS
                ;18196  LOOP.T2C.6:
U 163C, B610,15 ;18197          MOV LS[T8] TO WR[0]             ;ERROR NUMBER
U 163D, BE82,15 ;18198          MOV WR[0] TO LS[ERROR.NUMBER]   ;RESTORE ERROR NUMBER IN LS
U 163E, E58A,15 ;18199          CLR LS[ERROR.MASK]              ;CHECK ALL BITS AGAIN
                ;18200  LOOP.T2C.5:
U 163F, 189D,F5 ;18201          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;18202                                          ;SET UP TO READ MEMORY ADDRESS 0
U 1640, 3022,15 ;18203          MOV MEM.DATA TO WR[0]           ;READ ADDRESS 0
U 1641, B642,95 ;18204          MOV LS[#2] TO WR[1]             ;EXPECTED RESULT (UNCORRECTED DATA)
U 1642, 0869,3C ;18205          JSR [CHECK.RESULT]              ;VERIFY MEMORY ADDRESS 0 CONTAINS A 2
U 1643, 0963,F4 ;18206          JMP [LOOP.T2C.5]                ;ERROR LOOP IF ENABLED
                ;18207
```

```
U 1644, FF82,15 ;18208          INC LS[ERROR.NUMBER]              ;ERROR 6
U 1645, BE8B,15 ;18209          MOV WR[2] TO LS[ERROR.MASK]       ;CSR 1 MASK CHECKS ERROR BITS
U 1646, B67E,95 ;18210          MOV LS[RDS] TO WR[1]              ;EXPECTED DATA (RDS SET, OTHERS CLEAR)
U 1647, 9D45,75 ;18211          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;18212                                            ;SET UP TO READ CSR 1
U 1648, 3022,15 ;18213          MOV MEM.DATA TO WR[0]             ;READ CSR1
U 1649, 0869,3C ;18214          JSR [CHECK.RESULT]                ;CHECK FOR CORRECT CSR1 CONTENTS
U 164A, 0963,C4 ;18215          JMP [LOOP.T2C.6]                  ;ERROR LOOP IF ENABLED
                ;18216
U 164B, FF82,15 ;18217          INC LS[ERROR.NUMBER]              ;ERROR 7
U 164C, 0A17,EC ;18218          JSR [CLEAR.CSR1]                  ;CLEAR CSR
U 164D, 999C,75 ;18219          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;18220                                            ;SET UP TO WRITE AT LOCATION 0
U 164E, B29C,15 ;18221          WRITE.MEM LS[ZERO]                ;WRITE ALL 0'S IN MEMORY
U 164F, 3672,15 ;18222          MOV LS[ECC.DIS] TO WR[0]          ;SET ECC DISABLE BIT
U 1650, 8A17,FC ;18223          JSR [WRITE.CSR1]                  ;WRITE CSR 1 WITH DATA
U 1651, 3682,15 ;18224          MOV LS[ERROR.NUMBER] TO WR[0]     ;PUT IN WR
U 1652, 3E10,15 ;18225          MOV WR[0] TO LS[T8]               ;SAVE IN TEMP LS
                ;18226  LOOP.T2C.8.9:
U 1653, B610,15 ;18227          MOV LS[T8] TO WR[0]               ;ERROR NUMBER
U 1654, BE82,15 ;18228          MOV WR[0] TO LS[ERROR.NUMBER]     ;ERROR NUMBER RESTORED IN LS
U 1655, E58A,15 ;18229          CLR LS[ERROR.MASK]                ;CHECK ALL BITS
                ;18230  LOOP.T2C.7:
U 1656, 189D,F5 ;18231          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;18232                                            ;SET UP TO READ LOCATION 0
U 1657, 3022,15 ;18233          MOV MEM.DATA TO WR[0]             ;GET RESULT
U 1658, 2F82,95 ;18234          CLR WR[1]                         ;EXPECTED DATA
U 1659, 0869,3C ;18235          JSR [CHECK.RESULT]                ;CHECK FOR ALL 0'S
U 165A, 0965,64 ;18236          JMP [LOOP.T2C.7]                  ;ERROR LOOP IF ENABLED
                ;18237
U 165B, FF82,15 ;18238          INC LS[ERROR.NUMBER]              ;ERROR 8
U 165C, 3E8B,95 ;18239          MOV WR[3] TO LS[ERROR.MASK]       ;CHECK BITS 0-6 ONLY
U 165D, 9D9D,75 ;18240          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                ;18241                                            ;SET UP TO READ CSR 0
U 165E, 3022,15 ;18242          MOV MEM.DATA TO WR[0]             ;READ CSR 0
U 165F, 3700,95 ;18243          MOV LS[CKBTS.0111100] TO WR[1]    ;COMPLEMENT OF EXPECTED CKBTS
U 1660, A0C2,95 ;18244          COM WR[1]                         ;EXPECTED DATA
U 1661, 0869,3C ;18245          JSR [CHECK.RESULT]                ;SEE IF CORRECT CKBTS CAME BACK
U 1662, 0965,34 ;18246          JMP [LOOP.T2C.8.9]                ;ERROR LOOP IF ENABLED
                ;18247
U 1663, FF82,15 ;18248          INC LS[ERROR.NUMBER]              ;ERROR 9
U 1664, BE8B,15 ;18249          MOV WR[2] TO LS[ERROR.MASK]       ;CSR 1 ERROR MASK CHECKS ERROR BITS
U 1665, 9D45,75 ;18250          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;18251                                            ;SET UP TO READ CSR1
U 1666, 3022,15 ;18252          MOV MEM.DATA TO WR[0]             ;READ CSR 1
U 1667, 2F82,95 ;18253          CLR WR[1]                         ;EXPECTED DATA (ALL ERROR BITS CLEAR)
U 1668, 0869,3C ;18254          JSR [CHECK.RESULT]                ;SEE IF CSR 1 DATA IS CORRECT
U 1669, 0965,34 ;18255          JMP [LOOP.T2C.8.9]                ;ERROR LOOP IF ENABLED
                ;18256
U 166A, FF82,15 ;18257          INC LS[ERROR.NUMBER]              ;ERROR A
U 166B, 3704,15 ;18258          MOV LS[CKBTS.1100100] TO WR[0]    ;GET CKBTS FOR DATA OF 1 IN WR 0
U 166C, A0C0,15 ;18259          COM WR[0]                         ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 166D, 452C,15 ;18260          BIC LS[#FFFFFF00] TO WR[0]        ;CLEAR ALL BUT LOW BYTE
U 166E, 8A17,FC ;18261          JSR [WRITE.CSR1]                  ;PUT CKBTS FOR DATA OF 0 IN CSR 1, CLEAR
                ;18262                                            ; ALL OTHER BITS IN CSR
```

```
U 166F, 1D40,F5 ;18263          MEM.REQ[MAINT.ECC.DATA] ADRS[#1] DT[LONG]
                ;18264                                  ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;18265                                  ; SET FOR BRANCH
U 1670, B29C,15 ;18266          WRITE.MEM LS[ZERO]      ;WRITE ALL 0'S AT LOCATION 0 WITH
                ;18267                                  ; CKBTS FOR DATA OF A 1
U 1671, 3672,15 ;18268          MOV LS[ECC.DIS] TO WR[0] ;SET ECC DISABLE BIT
U 1672, 8A17,FC ;18269          JSR [WRITE.CSR1]        ;WRITE CSR 1 WITH DATA
U 1673, 3682,15 ;18270          MOV LS[ERROR.NUMBER] TO WR[0] ;PUT IN WR
U 1674, 3E10,15 ;18271          MOV WR[0] TO LS[T8]     ;SAVE IN TEMP LS
                ;18272  LOOP.T2C.B.C:
U 1675, B610,15 ;18273          MOV LS[T8] TO WR[0]     ;GET ERROR NUMBER
U 1676, BE82,15 ;18274          MOV WR[0] TO LS[ERROR.NUMBER] ;RESTORE ERROR NUMBER IN LS
U 1677, E58A,15 ;18275          CLR LS[ERROR.MASK]      ;CHECK ALL BITS
                ;18276  LOOP.T2C.A:
U 1678, 189D,F5 ;18277          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;18278                                  ;SET UP TO READ MEMORY LOCATION 0
U 1679, 3022,15 ;18279          MOV MEM.DATA TO WR[0]   ;READ ADDRESS 0
U 167A, 2F82,95 ;18280          CLR WR[1]               ;EXPECTED DATA
U 167B, 0869,3C ;18281          JSR [CHECK.RESULT]      ;SEE THAT NO CORRECTION OCCURRED
U 167C, 0967,84 ;18282          JMP [LOOP.T2C.A]        ;ERROR LOOP IF ENABLED
                ;18283
U 167D, FF82,15 ;18284          INC LS[ERROR.NUMBER]    ;ERROR B
U 167E, 3E8B,95 ;18285          MOV WR[3] TO LS[ERROR.MASK] ;CHECK BITS 0-6 ONLY
U 167F, 9D9D,75 ;18286          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                ;18287                                  ;SET UP TO READ CSR 0
U 1680, 3022,15 ;18288          MOV MEM.DATA TO WR[0]   ;READ CSR 0
U 1681, B704,95 ;18289          MOV LS[CKBTS.1100100] TO WR[1] ;COMPLEMENT OF EXPECTED CKBTS
U 1682, A0C2,95 ;18290          COM WR[1]               ;EXPECTED DATA
U 1683, 0869,3C ;18291          JSR [CHECK.RESULT]      ;SEE IF CORRECT CKBTS CAME BACK
U 1684, 8967,54 ;18292          JMP [LOOP.T2C.B.C]      ;ERROR LOOP IF ENABLED
                ;18293
U 1685, FF82,15 ;18294          INC LS[ERROR.NUMBER]    ;ERROR C
U 1686, BE8B,15 ;18295          MOV WR[2] TO LS[ERROR.MASK] ;CSR 1 ERROR MASK CHECKS ERROR BITS
U 1687, 9D45,75 ;18296          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;18297                                  ;SET UP TO READ CSR1
U 1688, 3022,15 ;18298          MOV MEM.DATA TO WR[0]   ;READ CSR 1
U 1689, 367C,95 ;18299          MOV LS[CRD] TO WR[1]    ;EXPECTED DATA (CRD SET)
U 168A, 0869,3C ;18300          JSR [CHECK.RESULT]      ;SEE IF CSR 1 DATA IS CORRECT
U 168B, 8967,54 ;18301          JMP [LOOP.T2C.B.C]      ;ERROR LOOP IF ENABLED
                ;18302
U 168C, FF82,15 ;18303          INC LS[ERROR.NUMBER]    ;ERROR D
U 168D, 0A17,EC ;18304          JSR [CLEAR.CSR1]        ;CLEAR CSR
U 168E, 999C,75 ;18305          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;18306                                  ;SET UP TO WRITE AT LOCATION 0
U 168F, 3240,15 ;18307          WRITE.MEM LS[#1]        ;WRITE A 1 IN MEMORY
                ;18308
U 1690, B700,15 ;18309          MOV LS[CKBTS.0111100] TO WR[0] ;GET CKBTS FOR DATA OF 0 IN WR 0
U 1691, A0C0,15 ;18310          COM WR[0]               ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 1692, 452C,15 ;18311          BIC LS[#FFFFFF00] TO WR[0] ;CLEAR ALL BUT LOW BYTE
U 1693, 4772,15 ;18312          BIS LS[ECC.DIS] TO WR[0] ;SET ECC DISABLE BIT
U 1694, 4774,15 ;18313          BIS LS[DIAG.CHK] TO WR[0] ;ALSO SET DIAG CHK MODE BIT
U 1695, 8A17,FC ;18314          JSR [WRITE.CSR1]        ;WRITE CSR 1 WITH DATA
U 1696, 3682,15 ;18315          MOV LS[ERROR.NUMBER] TO WR[0] ;PUT ERROR NUMBER IN WR
U 1697, 3E10,15 ;18316          MOV WR[0] TO LS[T8]     ;SAVE IN TEMP LS
                ;18317  LOOP.T2C.E.F:
```

```
U 1698, B610,15 ;18318          MOV LS[T8] TO WR[0]              ;ERROR NUMBER
U 1699, BE82,15 ;18319          MOV WR[0] TO LS[ERROR.NUMBER]    ;ERROR NUMBER RESTORED IN LS
U 169A, E58A,15 ;18320          CLR LS[ERROR.MASK]               ;CHECK ALL BITS
                ;18321  LOOP.T2C.D:
U 169B, 189D,F5 ;18322          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;18323                                           ;SET UP TO READ LOCATION 0
U 169C, 3022,15 ;18324          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 169D, 3640,95 ;18325          MOV LS[#1] TO WR[1]              ;EXPECTED DATA
U 169E, 0869,3C ;18326          JSR [CHECK.RESULT]               ;CHECK FOR ALL UNCORRECTED DATA
U 169F, 8969,B4 ;18327          JMP [LOOP.T2C.D]                 ;ERROR LOOP IF ENABLED
                ;18328
U 16A0, FF82,15 ;18329          INC LS[ERROR.NUMBER]             ;ERROR E
U 16A1, 3E8B,95 ;18330          MOV WR[3] TO LS[ERROR.MASK]      ;CHECK BITS 0-6 ONLY
U 16A2, 9D9D,75 ;18331          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                ;18332                                           ;SET UP TO READ CSR 0
U 16A3, 3022,15 ;18333          MOV MEM.DATA TO WR[0]            ;READ CSR 0
U 16A4, 3700,95 ;18334          MOV LS[CKBTS.0111100] TO WR[1]   ;COMPLEMENT OF EXPECTED CKBTS
U 16A5, A0C2,95 ;18335          COM WR[1]                        ;EXPECTED DATA
U 16A6, 0869,3C ;18336          JSR [CHECK.RESULT]               ;SEE IF CORRECT CKBTS CAME BACK
U 16A7, 8969,84 ;18337          JMP [LOOP.T2C.E.F]               ;ERROR LOOP IF ENABLED
                ;18338
U 16A8, FF82,15 ;18339          INC LS[ERROR.NUMBER]             ;ERROR F
U 16A9, BE8B,15 ;18340          MOV WR[2] TO LS[ERROR.MASK]      ;CSR 1 ERROR MASK CHECKS ERROR BITS
U 16AA, 9D45,75 ;18341          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;18342                                           ;SET UP TO READ CSR1
U 16AB, 3022,15 ;18343          MOV MEM.DATA TO WR[0]            ;READ CSR 1
U 16AC, 367C,95 ;18344          MOV LS[CRD] TO WR[1]             ;EXPECTED DATA (CRD SET)
U 16AD, 0869,3C ;18345          JSR [CHECK.RESULT]               ;SEE IF CSR 1 DATA IS CORRECT
U 16AE, 8969,84 ;18346          JMP [LOOP.T2C.E.F]               ;ERROR LOOP IF ENABLED
                ;18347
U 16AF, FF82,15 ;18348          INC LS[ERROR.NUMBER]             ;ERROR 10
U 16B0, 3704,15 ;18349          MOV LS[CKBTS.1100100] TO WR[0]   ;GET CKBTS FOR DATA OF 1 IN WR 0
U 16B1, A0C0,15 ;18350          COM WR[0]                        ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 16B2, 452C,15 ;18351          BIC LS[#FFFFFF00] TO WR[0]       ;CLEAR ALL BUT LOW BYTE
U 16B3, 4772,15 ;18352          BIS LS[ECC.DIS] TO WR[0]         ;SET ECC DISABLE BIT
U 16B4, 4774,15 ;18353          BIS LS[DIAG.CHK] TO WR[0]        ;ALSO SET DIAG CHK MODE BIT
U 16B5, 8A17,FC ;18354          JSR [WRITE.CSR1]                 ;WRITE CSR 1 WITH DATA
U 16B6, 3682,15 ;18355          MOV LS[ERROR.NUMBER] TO WR[0]    ;PUT ERROR NUMBER IN WR
U 16B7, 3E10,15 ;18356          MOV WR[0] TO LS[T8]              ;SAVE IN TEMP LS
                ;18357  LOOP.T2C.11:
U 16B8, B610,15 ;18358          MOV LS[T8] TO WR[0]              ;ERROR NUMBER
U 16B9, BE82,15 ;18359          MOV WR[0] TO LS[ERROR.NUMBER]    ;ERROR NUMBER RESTORED IN LS
U 16BA, 3E8B,95 ;18360          MOV WR[3] TO LS[ERROR.MASK]      ;CHECK BITS 0-6 ONLY
                ;18361  LOOP.T2C.10:
U 16BB, 189D,F5 ;18362          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;18363                                           ;SET UP TO READ LOCATION 0
U 16BC, 3022,15 ;18364          MOV MEM.DATA TO WR[0]            ;GET RESULT BUT DON'T CHECK
U 16BD, 9D9D,75 ;18365          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                ;18366                                           ;SET UP TO READ CSR 0
U 16BE, 3022,15 ;18367          MOV MEM.DATA TO WR[0]            ;READ CSR 0
U 16BF, B704,95 ;18368          MOV LS[CKBTS.1100100] TO WR[1]   ;COMPLEMENT OF EXPECTED CKBTS
U 16C0, A0C2,95 ;18369          COM WR[1]                        ;EXPECTED DATA
U 16C1, 0869,3C ;18370          JSR [CHECK.RESULT]               ;SEE IF CORRECT CKBTS CAME BACK
U 16C2, 096B,B4 ;18371          JMP [LOOP.T2C.10]                ;ERROR LOOP IF ENABLED
                ;18372
```

```
U 16C3, FF82,15 ;18373          INC LS[ERROR.NUMBER]              ;ERROR 11
U 16C4, BE8B,15 ;18374          MOV WR[2] TO LS[ERROR.MASK]       ;CSR 1 ERROR MASK CHECKS ERROR BITS
U 16C5, 9D45,75 ;18375          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;18376                                            ;SET UP TO READ CSR1
U 16C6, 3022,15 ;18377          MOV MEM.DATA TO WR[0]             ;READ CSR 1
U 16C7, 2F82,95 ;18378          CLR WR[1]                         ;EXPECTED DATA
U 16C8, 0869,3C ;18379          JSR [CHECK.RESULT]                ;SEE IF CSR 1 DATA IS CORRECT
U 16C9, 096B,84 ;18380          JMP [LOOP.T2C.11]                 ;ERROR LOOP IF ENABLED
                ;18381
                ;18382  END.T2C:
```

```
;18383  .page   "  TEST 2D READ.V Tests Without Unibus Activity  (M8391 MCT Module)"
;18384  ;++
;18385  ;
;18386  ; Test Description:
;18387  ;
;18388  ;       This test is designed to check for faults in the control store PROMs
;18389  ;       by verifying all paths of the micro-code.  This test checks the paths
;18390  ;       taken when a READ.V.NOCHK, READ.V.RCHK, READ.V.WCHK and READ.V.WCHK
;18391  ;       LOCK is executed without any UNIBUS activity.
;18392  ;         The hardware involved with the PROTECTION PROM has been previously
;18393  ;       tested.  This test will simply check that a READ VIRTUAL can be
;18394  ;       correctly executed with each of the entry points described above.  The
;18395  ;       test will write a 1 in TB address 0 so that reads to virtual address 0
;18396  ;       will actually read physical address 200.  An alternating data pattern
;18397  ;       will be written in physical address 0 so that if a read goes there,
;18398  ;       the error will be detected.  Data patterns used will be all 1's and
;18399  ;       all 0's.
;18400  ;
;18401  ; Assumptions:
;18402  ;
;18403  ;       It is assumed that all previous tests have run successfully, and
;18404  ;       that all hardware (other than MCT control store PROMs) is operating
;18405  ;       correctly.
;18406  ;
;18407  ; Test Steps:
;18408  ;
;18409  ;       1) Set up error mask, error number, and module number in LS (for
;18410  ;          error printouts) and clear previous error number in LS.  Also
;18411  ;          set MME in CSR 1 so memory management is enabled.
;18412  ;       2) Write a 1 in the address portion of TB location 0.
;18413  ;       3) Write an alternating pattern (55555555(H)) in memory location 0
;18414  ;          using WRITE.P and read with READ.P to verify the data.
;18415  ;       4) Write all 0's in physical location 200 using WRITE.P.  Read with
;18416  ;          READ.V.NOCHK at virtual address 0 and check result.
;18417  ;       5) Repeat step 4 with data of all 1's.
;18418  ;       6) Repeat steps 4 and 5 with READ.V.RCHK, READ.V.WCHK, and
;18419  ;          READ.V.WCHK.LOCKED.
;18420  ;
;18421  ; Errors:
;18422  ;
;18423  ;
;18424  ;       Error 1 - Initial write/read at physical address 0 failed.
;18425  ;       Error 2 - READ.V.NOCHK failed to read virtual address 0 (physical
;18426  ;                 address 200) correctly.  Data of all 0's.
;18427  ;       Error 3 - READ.V.NOCHK failed to read virtual address 0 (physical
;18428  ;                 address 200) correctly.  Data of all 1's.
;18429  ;       Error 4 - READ.V.RCHK failed to read virtual address 0 (physical
;18430  ;                 address 200) correctly.  Data of all 0's.
;18431  ;       Error 5 - READ.V.RCHK failed to read virtual address 0 (physical
;18432  ;                 address 200) correctly.  Data of all 1's.
;18433  ;       Error 6 - READ.V.WCHK failed to read virtual address 0 (physical
;18434  ;                 address 200) correctly.  Data of all 0's.
;18435  ;       Error 7 - READ.V.WCHK failed to read virtual address 0 (physical
;18436  ;                 address 200) correctly.  Data of all 1's.
;18437  ;       Error 8 - READ.V.WCHK.LOCKED failed to read virtual address 0
```

```
                    ;18438  ;                   (physical address 200) correctly.  Data of all 0's.
                    ;18439  ;       Error 9 - READ.V.WCHK.LOCKED failed to read virtual address 0
                    ;18440  ;                   (physical address 200) correctly.  Data of all 1's.
                    ;18441  ;
                    ;18442  ; Debug:
                    ;18443  ;
                    ;18444  ;       Note:  The following errors most likely indicate a problem with the
                    ;18445  ;       micro-code flow.  The hardware used by the micro-code has been pre-
                    ;18446  ;       viously tested.  To trace a problem, check the sequence outlined
                    ;18447  ;       below.  If the sequence is OK, suspect a bad bit in one of the control
                    ;18448  ;       store PROMs and check each state for the correct enables.
                    ;18449  ;
                    ;18450  ;       Error 1 - This error indicates a basic problem with the memory.  Run
                    ;18451  ;       previous tests again to pinpoint the problem.
                    ;18452  ;
                    ;18453  ;       Error 2 - This error indicates a problem with the micro flow in the
                    ;18454  ;       READ ARRAY micro code.  The correct flow is from C1 to CPU.RD.V.NOCHK
                    ;18455  ;       to ARY CYC C4.  The rest of the flow is identical to the READ.P flow
                    ;18456  ;       verified earlier.
                    ;18457  ;
                    ;18458  ;       Error 3 - Same as error 2 with all 1's data.
                    ;18459  ;
                    ;18460  ;       Error 4 - This error indicates a problem with the micro flow in the
                    ;18461  ;       READ ARRAY micro code.  The correct flow is from C1 to CPU.RD.V.RDCHK
                    ;18462  ;       to ARY CYC C4.  The rest of the flow is identical to the READ.P flow
                    ;18463  ;       verified earlier.
                    ;18464  ;
                    ;18465  ;       Error 5 - Same as error 4 with all 1's data.
                    ;18466  ;
                    ;18467  ;       Error 6 - This error indicates a problem with the micro flow in the
                    ;18468  ;       READ ARRAY micro code.  The correct flow is from C1 to CPU.RD.V.WRCHK
                    ;18469  ;       to ARY CYC C4.  The rest of the flow is identical to the READ.P flow
                    ;18470  ;       verified earlier.
                    ;18471  ;
                    ;18472  ;       Error 7 - Same as error 6 with all 1's data.
                    ;18473  ;
                    ;18474  ;       Error 8 - This error indicates a problem with the micro flow in the
                    ;18475  ;       READ ARRAY micro code.  The correct flow is from C1 to READ.V.WCHK
                    ;18476  ;       .LOCK TO READ.V.WCHK.LOCK.C4 to ARY CYC C4.  The rest of the flow is
                    ;18477  ;       identical to the READ.P flow verified earlier.
                    ;18478  ;
                    ;18479  ;       Error 9 - Same as error 8 with all 1's data.
                    ;18480  ;
                    ;18481  ;--
                    ;18482
                    ;18483  T.2D:
U 16CA, B65E,15     ;18484          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;18485                                          ; STATUS WORD
U 16CB, 3E80,15     ;18486          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;18487                                          ; BIT 15 INDICATES START OF TEST TO
                    ;18488                                          ; CONSOLE
U 16CC, 10E0,15     ;18489          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;18490  WAIT.T2D.0:
U 16CD, 896C,D4     ;18491          JMP [WAIT.T2D.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;18492                                          ; RESPOND
```

```
                    ;18493
U 16CE, 0A1A,AC     ;18494          JSR [SETUP.1]                       ;SET UP MASKS, MODULE CODE ETC.
                    ;18495
U 16CF, 0A17,DC     ;18496          JSR [WRITE.CSR1.MME]                ;SET MME IN CSR 1, CLEAR OTHER BITS
U 16D0, 989C,F5     ;18497          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                    ;18498                                              ;SET UP TO WRITE TB ADDRESS 0
U 16D1, 3240,15     ;18499          WRITE.MEM LS[#1]                    ;WRITE A 1 IN ADDRESS PORTION OF TB
                    ;18500  LOOP.T2D.1:
U 16D2, 999C,75     ;18501          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                    ;18502                                              ;SET UP TO WRITE MEMORY
U 16D3, B29A,15     ;18503          WRITE.MEM LS[#55555555]             ;WRITE ALTERNATING PATTERN AT ADDRESS 0
U 16D4, B69A,95     ;18504          MOV LS[#55555555] TO WR[1]          ;EXPECTED DATA
U 16D5, 189D,F5     ;18505          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                    ;18506                                              ;SET UP TO READ MEMORY
U 16D6, 3022,15     ;18507          MOV MEM.DATA TO WR[0]               ;READ DATA AT ADDRESS 0
U 16D7, 0869,3C     ;18508          JSR [CHECK.RESULT]                  ;CHECK FOR ERROR
U 16D8, 096D,24     ;18509          JMP [LOOP.T2D.1]                    ;ERROR LOOP IF ENABLED
                    ;18510
U 16D9, FF82,15     ;18511          INC LS[ERROR.NUMBER]                ;ERROR 2
U 16DA, 1952,75     ;18512          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                    ;18513                                              ;SET UP TO WRITE MEMORY ADDRESS 200
U 16DB, B29C,15     ;18514          WRITE.MEM LS[ZERO]                  ;WRITE DATA OF ALL 0'S AT 200
                    ;18515  LOOP.T2D.2:
U 16DC, 2F82,95     ;18516          CLR WR[1]                           ;EXPECTED DATA
U 16DD, 9A9C,75     ;18517          MEM.REQ[READ.V.NOCHK] ADRS[#0] DT[LONG]
                    ;18518                                              ;SET UP TO READ VIRTUAL ADDRESS 0
U 16DE, 3022,15     ;18519          MOV MEM.DATA TO WR[0]               ;READ VIR 0 (PHY 200)
U 16DF, 0869,3C     ;18520          JSR [CHECK.RESULT]                  ;CHECK FOR CORRECT DATA
U 16E0, 896D,C4     ;18521          JMP [LOOP.T2D.2]                    ;ERROR LOOP IF ENABLED
                    ;18522
U 16E1, FF82,15     ;18523          INC LS[ERROR.NUMBER]                ;ERROR 3
U 16E2, 1952,75     ;18524          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                    ;18525                                              ;SET UP TO WRITE MEMORY ADDRESS 200
U 16E3, 329E,15     ;18526          WRITE.MEM LS[ONES]                  ;WRITE DATA OF ALL 1'S AT 200
                    ;18527  LOOP.T2D.3:
U 16E4, 369E,95     ;18528          MOV LS[ONES] TO WR[1]               ;EXPECTED DATA
U 16E5, 9A9C,75     ;18529          MEM.REQ[READ.V.NOCHK] ADRS[#0] DT[LONG]
                    ;18530                                              ;SET UP TO READ VIRTUAL ADDRESS 0
U 16E6, 3022,15     ;18531          MOV MEM.DATA TO WR[0]               ;READ VIR 0 (PHY 200)
U 16E7, 0869,3C     ;18532          JSR [CHECK.RESULT]                  ;CHECK FOR CORRECT DATA
U 16E8, 096E,44     ;18533          JMP [LOOP.T2D.3]                    ;ERROR LOOP IF ENABLED
                    ;18534
U 16E9, FF82,15     ;18535          INC LS[ERROR.NUMBER]                ;ERROR 4
U 16EA, 1952,75     ;18536          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                    ;18537                                              ;SET UP TO WRITE MEMORY ADDRESS 200
U 16EB, B29C,15     ;18538          WRITE.MEM LS[ZERO]                  ;WRITE DATA OF ALL 0'S AT 200
                    ;18539  LOOP.T2D.4:
U 16EC, 2F82,95     ;18540          CLR WR[1]                           ;EXPECTED DATA
U 16ED, 1A9D,75     ;18541          MEM.REQ[READ.V.RCHK] ADRS[#0] DT[LONG]
                    ;18542                                              ;SET UP TO READ VIRTUAL ADDRESS 0
U 16EE, 3022,15     ;18543          MOV MEM.DATA TO WR[0]               ;READ VIR 0 (PHY 200)
U 16EF, 0869,3C     ;18544          JSR [CHECK.RESULT]                  ;CHECK FOR CORRECT DATA
U 16F0, 896E,C4     ;18545          JMP [LOOP.T2D.4]                    ;ERROR LOOP IF ENABLED
                    ;18546
U 16F1, FF82,15     ;18547          INC LS[ERROR.NUMBER]                ;ERROR 5
```

```
U 16F2, 1952,75 ;18548          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                ;18549                                  ;SET UP TO WRITE MEMORY ADDRESS 200
U 16F3, 329E,15 ;18550          WRITE.MEM LS[ONES]      ;WRITE DATA OF ALL 1'S AT 200
                ;18551  LOOP.T2D.5:
U 16F4, 369E,95 ;18552          MOV LS[ONES] TO WR[1]   ;EXPECTED DATA
U 16F5, 1A9D,75 ;18553          MEM.REQ[READ.V.RCHK] ADRS[#0] DT[LONG]
                ;18554                                  ;SET UP TO READ VIRTUAL ADDRESS 0
U 16F6, 3022,15 ;18555          MOV MEM.DATA TO WR[0]   ;READ VIR 0 (PHY 200)
U 16F7, 0869,3C ;18556          JSR [CHECK.RESULT]      ;CHECK FOR CORRECT DATA
U 16F8, 896F,44 ;18557          JMP [LOOP.T2D.5]        ;ERROR LOOP IF ENABLED
                ;18558
U 16F9, FF82,15 ;18559          INC LS[ERROR.NUMBER]    ;ERROR 6
U 16FA, 1952,75 ;18560          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                ;18561                                  ;SET UP TO WRITE MEMORY ADDRESS 200
U 16FB, B29C,15 ;18562          WRITE.MEM LS[ZERO]      ;WRITE DATA OF ALL 0'S AT 200
                ;18563  LOOP.T2D.6:
U 16FC, 2F82,95 ;18564          CLR WR[1]               ;EXPECTED DATA
U 16FD, 1A9C,F5 ;18565          MEM.REQ[READ.V.WCHK] ADRS[#0] DT[LONG]
                ;18566                                  ;SET UP TO READ VIRTUAL ADDRESS 0
U 16FE, 3022,15 ;18567          MOV MEM.DATA TO WR[0]   ;READ VIR 0 (PHY 200)
U 16FF, 0869,3C ;18568          JSR [CHECK.RESULT]      ;CHECK FOR CORRECT DATA
U 1700, 096F,C4 ;18569          JMP [LOOP.T2D.6]        ;ERROR LOOP IF ENABLED
                ;18570
U 1701, FF82,15 ;18571          INC LS[ERROR.NUMBER]    ;ERROR 7
U 1702, 1952,75 ;18572          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                ;18573                                  ;SET UP TO WRITE MEMORY ADDRESS 200
U 1703, 329E,15 ;18574          WRITE.MEM LS[ONES]      ;WRITE DATA OF ALL 1'S AT 200
                ;18575  LOOP.T2D.7:
U 1704, 369E,95 ;18576          MOV LS[ONES] TO WR[1]   ;EXPECTED DATA
U 1705, 1A9C,F5 ;18577          MEM.REQ[READ.V.WCHK] ADRS[#0] DT[LONG]
                ;18578                                  ;SET UP TO READ VIRTUAL ADDRESS 0
U 1706, 3022,15 ;18579          MOV MEM.DATA TO WR[0]   ;READ VIR 0 (PHY 200)
U 1707, 0869,3C ;18580          JSR [CHECK.RESULT]      ;CHECK FOR CORRECT DATA
U 1708, 0970,44 ;18581          JMP [LOOP.T2D.7]        ;ERROR LOOP IF ENABLED
                ;18582
U 1709, FF82,15 ;18583          INC LS[ERROR.NUMBER]    ;ERROR 8
U 170A, 1952,75 ;18584          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                ;18585                                  ;SET UP TO WRITE MEMORY ADDRESS 200
U 170B, B29C,15 ;18586          WRITE.MEM LS[ZERO]      ;WRITE DATA OF ALL 0'S AT 200
                ;18587  LOOP.T2D.8:
U 170C, 2F82,95 ;18588          CLR WR[1]               ;EXPECTED DATA
U 170D, 9E9C,F5 ;18589          MEM.REQ[READ.V.WCHK.LOCKED] ADRS[#0] DT[LONG]
                ;18590                                  ;SET UP TO READ VIRTUAL ADDRESS 0
U 170E, 3022,15 ;18591          MOV MEM.DATA TO WR[0]   ;READ VIR 0 (PHY 200)
U 170F, 0869,3C ;18592          JSR [CHECK.RESULT]      ;CHECK FOR CORRECT DATA
U 1710, 8970,C4 ;18593          JMP [LOOP.T2D.8]        ;ERROR LOOP IF ENABLED
                ;18594
U 1711, FF82,15 ;18595          INC LS[ERROR.NUMBER]    ;ERROR 9
U 1712, 1952,75 ;18596          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                ;18597                                  ;SET UP TO WRITE MEMORY ADDRESS 200
U 1713, 329E,15 ;18598          WRITE.MEM LS[ONES]      ;WRITE DATA OF ALL 1'S AT 200
                ;18599  LOOP.T2D.9:
U 1714, 369E,95 ;18600          MOV LS[ONES] TO WR[1]   ;EXPECTED DATA
U 1715, 9E9C,F5 ;18601          MEM.REQ[READ.V.WCHK.LOCKED] ADRS[#0] DT[LONG]
                ;18602                                  ;SET UP TO READ VIRTUAL ADDRESS 0
```

```
U 1716, 3022,15 ;18603          MOV MEM.DATA TO WR[0]          ;READ VIR 0 (PHY 200)
U 1717, 0869,3C ;18604          JSR [CHECK.RESULT]             ;CHECK FOR CORRECT DATA
U 1718, 8971,44 ;18605          JMP [LOOP.T2D.9]               ;ERROR LOOP IF ENABLED
                ;18606
                ;18607  END.T2D:
```

```
;18608  .page   "  TEST 2E READ.V.RCHK.IFILL Flow with Error (M8391 MCT Module)"
;18609  ;++
;18610  ;
;18611  ; Test Description:
;18612  ;
;18613  ;       This test is designed to check for faults in the control store PROMs
;18614  ;       by verifying all paths of the micro-code.  This test checks the path
;18615  ;       taken when a single bit error or an error sum occurs on an IB fill.
;18616  ;       The test writes a single bit error in memory by using the MAINT
;18617  ;       .ECC.DATA routine.  CKBTS for all 0's are written with data of 1 to
;18618  ;       produce a single bit error at location 0.  Then a READ.V.RCHK.IFILL
;18619  ;       is performed at address 0.  A MOV IB.DATA TO OS is executed with PC=0
;18620  ;       and the OS is checked for corrected (all 0's) data.  CSR 1 is checked
;18621  ;       for the CRD bit set and CSR 0 is checked for the correct syndrome bits.
;18622  ;       Then all ones are written in memory with correct CKBTs.  The TB PAR
;18623  ;       DIAG bit is set in CSR 1 and the test is repeated.  The resulting error
;18624  ;       sum should prevent the IB fill and all 0's should again be read.  CSR 1
;18625  ;       is checked to assure that a TB MISS occured.
;18626  ;
;18627  ; Assumptions:
;18628  ;
;18629  ;       It is assumed that all previous tests have run successfully, and
;18630  ;       that all hardware (other than MCT control store PROMs) is operating
;18631  ;       correctly.
;18632  ;
;18633  ; Test Steps:
;18634  ;
;18635  ;       1)  Set up error mask, error number, and module number in LS (for
;18636  ;           error printouts) and clear previous error number in LS.
;18637  ;       2)  Load CSR 1 with ckbts of 0111100(B) (these must be inverted before
;18638  ;           writing in CSR 1).  These are ckbts for data of all 0's.  Clear
;18639  ;           other bits of CSR 1.
;18640  ;       3)  Issue a MEM.REQ with MF=MAINT.ECC.DATA and DT=LONGWORD.  Use an
;18641  ;           address of 1 (LVA00 set) to write address 0 (LVA00 is used as a
;18642  ;           branch condition in the MAINT.ECC.DATA flow).Then issue a WRITE.MEM
;18643  ;           LS with data of a 1.
;18644  ;       4)  Write TB address 0 with a 1 in the VALID bit position so that
;18645  ;           virtual address 0 corresponds to physical address 0.
;18646  ;       5)  Set the MME bit in CSR 1 and execute a READ.V.RCHK.IBFILL at
;18647  ;           address 0.  Then execute a MOV IB.DATA TO OS to get the result out
;18648  ;           of the IB.
;18649  ;       6)  Check OS for all 0's (corrected data).  Then check CSR 1 for CRD
;18650  ;           set and CSR 0 for the correct syndrome bits.
;18651  ;       7)  Write all ones at address 0.  Repeat step 5 with the TB PAR DIAG
;18652  ;           bit in CSR 1 also set.  Check OS for no change (data not modified
;18653  ;           on error sum) and check CSR 1 for TB MISS set.
;18654  ;
;18655  ; Errors:
;18656  ;
;18657  ;       NOTE:  Expected and received data is as follows:  IB data when error
;18658  ;              mask is FFFFFF00, CSR 1 data when mask is 29003FFF, and CSR 0
;18659  ;              when error mask is FFFFFF80.
;18660  ;
;18661  ;       Error 1 - Single bit error not corrected correctly.
;18662  ;       Error 2 - CSR 1 CRD not set or other bits set.
```

```
                ;18663  ;       Error 3 - CSR 0 syndrome bits incorrect.
                ;18664  ;       Error 4 - IB fill not aborted on error sum.
                ;18665  ;       Error 5 - CSR 1 TB MISS not set or other bits set.
                ;18666  ;
                ;18667  ; Debug:
                ;18668  ;
                ;18669  ;       Note:  The following errors most likely indicate a problem with the
                ;18670  ;       micro-code flow.  The hardware used by the micro-code has been pre-
                ;18671  ;       viously tested.  To trace a problem, check the sequence outlined
                ;18672  ;       below.  If the sequence is OK, suspect a bad bit in one of the control
                ;18673  ;       store PROMs and check each state for the correct enables.
                ;18674  ;
                ;18675  ;       Error 1 - This error indicates a problem with the micro flow as-
                ;18676  ;       sociated with the SERR branch in the READ.V.RCHK.IFILL micro code.
                ;18677  ;       The correct sequence of micro instructions from CPU.ISTREAM.REQ.C6
                ;18678  ;       follows:  The ERRSUM NOT and CPUG NOT branch is taken.  From there
                ;18679  ;       the ERR path should be taken to READ.V.RCHK.IFILL D.E.  The SERR path
                ;18680  ;       should be taken next to READ.V.RCHK.IFILL D.E. C1 to READ.V.RCHK.IFILL
                ;18681  ;       D.E. C2 to READ.V.RCHK.IFILL D.E. C3 to READ.V.RCHK.IFILL C8 to IDLE.
                ;18682  ;
                ;18683  ;       Error 2 and 3 - Same as error 1.  Suspect a problem with the SERR path.
                ;18684  ;
                ;18685  ;       Error 4 - This error indicates a problem with the ERRSUM path.  From
                ;18686  ;       CPU.ISTREAM.REQ C6 the path taken should be the ERRSUM and CPUG NOT
                ;18687  ;       branch.
                ;18688  ;
                ;18689  ;       Error 5 - Suspect a problem with READ.V.RCHK IFILL C5 as this is the
                ;18690  ;       micro word that clocks the CSR error bits for ERRSUM.
                ;18691  ;
                ;18692  ;--
                ;18693
                ;18694  T.2E:
U 1719, B65E,15 ;18695          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;18696                                          ; STATUS WORD
U 171A, 3E80,15 ;18697  ,       MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;18698                                          ; BIT 15 INDICATES START OF TEST TO
                ;18699                                          ; CONSOLE
U 171B, 10E0,15 ;18700          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;18701  WAIT.T2E.0:
U 171C, 0971,C4 ;18702          JMP [WAIT.T2E.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;18703                                          ; RESPOND
                ;18704
U 171D, 0A1A,9C ;18705          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                ;18706                                          ; CLEAR MCT CSR 1
U 171E, B713,95 ;18707          MOV LS[#FFFFFF80] TO WR[3]      ;SET BITS 7-31
U 171F, B62F,15 ;18708          MOV LS[CSR1.MASK] TO WR[2]      ;ERROR MASK FOR ERROR 3.  DO NOT CHECK
                ;18709                                          ; BITS 0-13 OR BIT 24
U 1720, 4777,15 ;18710          BIS LS[MME] TO WR[2]            ;DON'T CHECK MME BIT
U 1721, 4778,15 ;18711          BIS LS[TB.PAR.DIAG] TO WR[2]    ;OR TB PARITY DIAG BIT
                ;18712
U 1722, 999C,75 ;18713          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;18714                                          ;SET UP TO WRITE MEMORY ADDRESS 0
U 1723, 329E,15 ;18715          WRITE.MEM LS[ONES]              ;WRITE ALL 1'S IN ADDRESS 0
U 1724, 9B9D,75 ;18716          MEM.REQ[READ.V.RCHK.IFILL] ADRS[#0] DT[LONG] .
                ;18717                                          ;SET UP TO FILL IB FROM MEMORY ADDRESS 0
```

```
                    ;18718
U 1725, 367E,15     ;18719          MOV LS[BIT31] TO WR[0]           ;SET BIT 31 IN WR
U 1726, C77A,15     ;18720          BIS LS[BIT29] TO WR[0]           ;AND BIT 29
U 1727, 4772,15     ;18721          BIS LS[BIT25] TO WR[0]           ;AND BIT 25
U 1728, BE12,15     ;18722          MOV WR[0] TO LS[T9]              ;PUT IN TEMP LS
U 1729, 989C,F5     ;18723          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                    ;18724                                           ;SET UP TO WRITE TB ADDRESS 0
U 172A, B212,15     ;18725          WRITE.MEM LS[T9]                 ;SET VALID, BYTE OFFSET AND PROTC BITS
                    ;18726                                           ; AND MAP VIRTUAL ADDRESS 0 TO PHYSICAL
                    ;18727                                           ; ADDRESS 0 IN TB
U 172B, B700,15     ;18728          MOV LS[CKBTS.0111100] TO WR[0]   ;GET CKBTS FOR DATA OF 0 IN WR 0
U 172C, A0C0,15     ;18729          COM WR[0]                        ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 172D, 452C,15     ;18730          BIC LS[#FFFFFF00] TO WR[0]       ;CLEAR ALL BUT LOW BYTE
U 172E, 8A17,FC     ;18731          JSR [WRITE.CSR1]                 ;PUT CKBTS FOR DATA OF 0 IN CSR 1, CLEAR
                    ;18732                                           ; ALL OTHER BITS IN CSR
U 172F, 1D40,F5     ;18733          MEM.REQ[MAINT.ECC.DATA] ADRS[#1] DT[LONG]
                    ;18734                                           ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                    ;18735                                           ; SET FOR BRANCH
U 1730, 3240,15     ;18736          WRITE.MEM LS[#1]                 ;WRITE A 1 AT LOCATION 0 WITH CKBTS FOR
                    ;18737                                           ; DATA OF ALL 0'S
U 1731, 0A17,DC     ;18738          JSR [WRITE.CSR1.MME]             ;WRITE IN CSR 1
                    ;18739  LOOP.T2E.2.3:
U 1732, B640,15     ;18740          MOV LS[#1] TO WR[0]              ;1 IN WR
U 1733, BE82,15     ;18741          MOV WR[0] TO LS[ERROR.NUMBER]    ;SET UP ERROR NUMBER IN LS
U 1734, B62C,15     ;18742          MOV LS[#FFFFFF00] TO WR[0]       ;SET UP ERROR MASK
U 1735, 3E8A,15     ;18743          MOV WR[0] TO LS[ERROR.MASK]      ;ERROR MASK TO CHECK LOW BYTE ONLY
                    ;18744  LOOP.T2E.1:
U 1736, 2F82,95     ;18745          CLR WR[1]                        ;EXPECTED DATA
U 1737, E520,15     ;18746          CLR LS[PC]                       ;BYTE IN IB REG TO LOAD INTO OS
U 1738, 9B9D,75     ;18747          MEM.REQ[READ.V.RCHK.IFILL] ADRS[#0] DT[LONG]
                    ;18748                                           ;SET UP TO FILL IB FROM MEMORY ADDRESS 0
U 1739, 8401,4E     ;18749          MOV IB.DATA TO OS                ;GET RESULT
U 173A, DB00,15     ;18750          NOP                              ;SKIP ON IB VALID WILL SKIP THIS
U 173B, B6F8,15     ;18751          MOV LS[OS] TO WR[0]              ;GET RESULT FROM IB REG
U 173C, 0869,3C     ;18752          JSR [CHECK.RESULT]               ;CHECK FOR DATA OF 0
U 173D, 8973,64     ;18753          JMP [LOOP.T2E.1]                 ;ERROR LOOP IF ENABLED
                    ;18754
U 173E, FF82,15     ;18755          INC LS[ERROR.NUMBER]             ;ERROR 2
U 173F, BE8B,15     ;18756          MOV WR[2] TO LS[ERROR.MASK]      ;ERROR MASK FOR CHECKING CSR 1
U 1740, 367C,95     ;18757          MOV LS[CRD] TO WR[1]             ;EXPECTED DATA (CRD SET, OTHERS CLEAR)
U 1741, 9D45,75     ;18758          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                    ;18759                                           ;READ CSR1
U 1742, 3022,15     ;18760          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 1743, 0869,3C     ;18761          JSR [CHECK.RESULT]               ;CHECK FOR ONLY CRD SET
U 1744, 0973,24     ;18762          JMP [LOOP.T2E.2.3]               ;ERROR LOOP IF ENABLED
                    ;18763
U 1745, FF82,15     ;18764          INC LS[ERROR.NUMBER]             ;ERROR 3
U 1746, 3E8B,95     ;18765          MOV WR[3] TO LS[ERROR.MASK]      ;ERROR MASK FOR CHECKING CSR 0
U 1747, 3646,95     ;18766          MOV LS[#8] TO WR[1]              ;SET BIT 3
U 1748, C748,95     ;18767          BIS LS[#10] TO WR[1]             ;AND BIT 4
U 1749, 474C,95     ;18768          BIS LS[#40] TO WR[1]             ;AND BIT 6 (1011000(B))
U 174A, A0C2,95     ;18769          COM WR[1]                        ;EXPECTED SYNDROMES
U 174B, 9D9D,75     ;18770          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
                    ;18771                                           ;READ CSR 0
U 174C, 3022,15     ;18772          MOV MEM.DATA TO WR[0]            ;GET RESULT
```

```
U 174D, 0869.3C ;18773          JSR [CHECK.RESULT]                  ;CHECK FOR CORRECT SYNDROMES
U 174E, 0973.24 ;18774          JMP [LOOP.T2E.2.3]                  ;ERROR LOOP IF ENABLED
                ;18775
U 174F, 999C.75 ;18776          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;18777                                              ;SET UP TO WRITE MEMORY ADDRESS 0
U 1750, 329E.15 ;18778          WRITE.MEM LS[ONES]                  ;WRITE ALL 1'S IN ADDRESS 0
U 1751, B676.15 ;18779          MOV LS[MME] TO WR[0]                ;SET MME BIT IN WR
U 1752, C77A.15 ;18780          BIS LS[TB.PAR.DIAG] TO WR[0]        ;ALSO SET TB PAR ERR DIAG BIT
U 1753, 8A17.FC ;18781          JSR [WRITE.CSR1]                    ;WRITE INTO CSR 1
U 1754, B62D.95 ;18782          MOV LS[#FFFFFF00] TO WR[3]          ;SET UP ERROR MASK FOR LOW BYTE IN WR[3]
                ;18783  LOOP.T2E.5:
U 1755, 3644.15 ;18784          MOV LS[#4] TO WR[0]                 ;4 IN WR
U 1756, BE82.15 ;18785          MOV WR[0] TO LS[ERROR.NUMBER]       ;SET UP ERROR NUMBER IN LS
U 1757, 3E8B.95 ;18786          MOV WR[3] TO LS[ERROR.MASK]         ;SET UP ERROR MASK TO CHECK LOW BYTE
                ;18787  LOOP.T2E.4:
U 1758, 2F82.95 ;18788          CLR WR[1]                           ;EXPECTED DATA
U 1759, E520.15 ;18789          CLR LS[PC]                          ;BYTE IN IB REG TO LOAD INTO OS
U 175A, 9B9D.75 ;18790          MEM.REQ[READ.V.RCHK.IFILL] ADRS[#0] DT[LONG]
                ;18791                                              ;SET UP TO FILL IB FROM MEMORY ADDRESS 0
U 175B, 8401.4E ;18792          MOV IB.DATA TO OS                   ;GET RESULT
U 175C, DB00.15 ;18793          NOP                                 ;SKIP ON IB VALID WILL SKIP THIS
U 175D, B6F8.15 ;18794          MOV LS[OS] TO WR[0]                 ;GET RESULT FROM IB REG
U 175E, 0869.3C ;18795          JSR [CHECK.RESULT]                  ;CHECK FOR DATA OF 0
U 175F, 0975.84 ;18796          JMP [LOOP.T2E.4]                    ;ERROR LOOP IF ENABLED
                ;18797
U 1760, FF82.15 ;18798          INC LS[ERROR.NUMBER]                ;ERROR 5
U 1761, BE8B.15 ;18799          MOV WR[2] TO LS[ERROR.MASK]         ;ERROR MASK FOR CHECKING CSR 1
U 1762, 365E.95 ;18800          MOV LS[TB.PAR.ERR] TO WR[1]         ;EXPECTED DATA (TB MISS SET, OTHERS CLEAR)
U 1763, 9D45.75 ;18801          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;18802                                              ;READ CSR1
U 1764, 3022.15 ;18803          MOV MEM.DATA TO WR[0]               ;GET RESULT
U 1765, 0869.3C ;18804          JSR [CHECK.RESULT]                  ;CHECK FOR ONLY TB MISS SET
U 1766, 8975.54 ;18805          JMP [LOOP.T2E.5]                    ;ERROR LOOP IF ENABLED
                ;18806
                ;18807  END.T2E:
```

```
;18808  .page   "  TEST 2F WRITE.V.WCHK Flow and Abort Write (M8391 MCT Module)"
;18809  ;++
;18810  ;
;18811  ; Test Description:
;18812  ;
;18813  ;       This test is designed to check for faults in the control store PROMs
;18814  ;       by verifying all paths of the micro-code.  This test checks the path
;18815  ;       taken when a WRITE.V.WCHK is issued for aligned longword and unaligned
;18816  ;       longword writes.  Also checked is the abort write on error sum path.
;18817  ;       The TB is written to map virtual address 0 to physical address 200 and
;18818  ;       the MME bit is set in CSR 1.  A WRITE.P to address 200 of all 1's is
;18819  ;       executed followed by an aligned write to virtual address 0 with all 0's
;18820  ;       data.  The data is checked via the READ.P flow.  An unaligned write is
;18821  ;       checked next by writing 1's at virtual address 2 and checking for a
;18822  ;       word of zeros and a word of ones at physical address 200.  Finally an
;18823  ;       aligned longword of 0's is attempted at virtual address 0 with the TB
;18824  ;       PAR DIAG bit set in CSR 1 to force an error sum.  Location 200 is
;18825  ;       checked to assure that the write was aborted.
;18826  ;
;18827  ; Assumptions:
;18828  ;
;18829  ;       It is assummed that all previous tests have run successfully, and
;18830  ;       that all hardware (other than MCT control store PROMs) is operating
;18831  ;       correctly.
;18832  ;
;18833  ; Test Steps:
;18834  ;
;18835  ;       1)  Set up error mask, error number, and module number in LS (for
;18836  ;           error printouts) and clear previous error number in LS.
;18837  ;       2)  Clear CSR 1.  Write all 1's at physical address 200 and write all
;18838  ;           0's at physical address 204.
;18839  ;       3)  Write TB address 0 with VALID, PROT C, MODIFY, and BYTE OFFSET set,
;18840  ;           and PA 09 set so virtual 0 maps to physical 200.
;18841  ;       4)  Set MME in CSR 1.
;18842  ;       5)  Execute WRITE.V.WCHK with data of all 0's to virtual address 0 and
;18843  ;           read result with READ.P at address 200.  Check for all 0's.
;18844  ;       6)  Execute WRITE.V.WCHK with data of all 1's to virtual address 2
;18845  ;           (unaligned write to physical address 202).  Read result with READ.P
;18846  ;           at address 200 and check for FFFF0000(H).
;18847  ;       7)  Set TB PAR ERR in CSR 1 and execute WRITE.V.WCHK with data of all
;18848  ;           zeros at virtual address 0.  Read physical address 200 and check
;18849  ;           that data was not modified (ERROR SUM should abort write).
;18850  ;
;18851  ; Errors:
;18852  ;
;18853  ;       Error 1 - WRITE.V.WCHK failed to write at virtual address 0 (physical
;18854  ;                 address 200).
;18855  ;       Error 2 - WRITE.V.WCHK failed to write unaligned longword at virtual
;18856  ;                 address 2 (physical address 202).
;18857  ;       Error 3 - WRITE.V.WCHK did not abort write on error sum (TB parity
;18858  ;                 error).
;18859  ;
;18860  ; Debug:
;18861  ;
;18862  ;       Note:  The following errors most likely indicate a problem with the
```

```
                ;18863  ;       micro-code flow.  The hardware used by the micro-code has been pre-
                ;18864  ;       viously tested.  To trace a problem, check the sequence outlined
                ;18865  ;       below.  If the sequence is OK, suspect a bad bit in one of the control
                ;18866  ;       store PROMs and check each state for the correct enables.
                ;18867  ;
                ;18868  ;       Error 1 - This error indicates a problem with the micro flow as-
                ;18869  ;       sociated with WRITE.V.WCHK.  The correct sequence of instructions
                ;18870  ;       is from WRITE.V.WCHK to WR ARY WCHK C4 to WR ARY WCHK C5 to WR ARY
                ;18871  ;       ALIGN LW C6 to WR ARY ALIGN LW C7.  The states after WR ARY ALIGN LW
                ;18872  ;       C7 have been checked previously.
                ;18873  ;
                ;18874  ;       Error 2 - This error probably indicates a problem with the ALIGN LW
                ;18875  ;       branch at WR ARY WCHK C5.  The flow is the same as above except that
                ;18876  ;       after WR ARY WCHK C5 the flow should go to WR ARY C6 and then to WR
                ;18877  ;       ARY C7.  The states following this have been checked previously.
                ;18878  ;
                ;18879  ;       Error 3 - This error indicates a problem with the ERRSUM branch at
                ;18880  ;       WR ARY ALIGN LW C6 or beyond.  The flow is the same as ERROR 1 up to
                ;18881  ;       WR ARY ALIGN LW C7.  From there the flow should go to ABORT WR CYC C1
                ;18882  ;       to ABORT WR WAIT to UB PREP and IDLE.
                ;18883  ;
                ;18884  ;--
                ;18885
                ;18886  T.2F:
U 1767, B65E,15 ;18887          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;18888                                          ; STATUS WORD
U 1768, 3E80,15 ;18889          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;18890                                          ; BIT 15 INDICATES START OF TEST TO
                ;18891                                          ; CONSOLE
U 1769, 10E0,15 ;18892          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;18893  WAIT.T2F.0:
U 176A, 8976,A4 ;18894          JMP [WAIT.T2F.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;18895                                          ; RESPOND
                ;18896
U 176B, 0A1A,9C ;18897          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                ;18898                                          ; CLEAR MCT CSR 1
                ;18899
U 176C, 1952,75 ;18900          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                ;18901                                          ;SET UP TO WRITE MEMORY ADDRESS 200
U 176D, 329E,15 ;18902          WRITE.MEM LS[ONES]              ;WRITE ALL 1'S IN ADDRESS 200
U 176E, B652,15 ;18903          MOV LS[#200] TO WR[0]           ;200 IN WR
U 176F, 4744,15 ;18904          BIS LS[#4] TO WR[0]             ;204 IN WR
U 1770, BE12,15 ;18905          MOV WR[0] TO LS[T9]             ;SET UP ADDRESS IN TEMPORARY LS LOCATION
U 1771, 9912,75 ;18906          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;18907                                          ;SET UP TO WRITE MEMORY ADDRESS 204
U 1772, B29C,15 ;18908          WRITE.MEM LS[ZERO]              ;WRITE ALL 0'S IN ADDRESS 204
                ;18909
U 1773, 367E,15 ;18910          MOV LS[BIT31] TO WR[0]          ;SET BIT 31 IN WR
U 1774, C77A,15 ;18911          BIS LS[BIT29] TO WR[0]          ;AND BIT 29
U 1775, 4774,15 ;18912          BIS LS[BIT26] TO WR[0]          ;AND BIT 26
U 1776, 4772,15 ;18913          BIS LS[BIT25] TO WR[0]          ;AND BIT 25
U 1777, C740,15 ;18914          BIS LS[BIT0] TO WR[0]           ;AND BIT 0
U 1778, BE12,15 ;18915          MOV WR[0] TO LS[T9]             ;PUT IN TEMP LS
U 1779, 989C,F5 ;18916          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;18917                                          ;SET UP TO WRITE TB ADDRESS 0
```

```
U 177A, B212,15 ;18918          WRITE.MEM LS[T9]                    ;SET VALID,PROT C,MODIFY,AND BYTE OFFSET
                ;18919                                              ; BITS AND MAP VIRTUAL ADDRESS 0 TO
                ;18920                                              ; PHYSICAL ADDRESS 200 IN TB
U 177B, 0A17,DC ;18921          JSR [WRITE.CSR1.MME]                ;WRITE IN CSR 1
                ;18922  LOOP.T2F.1:
U 177C, 2F82,95 ;18923          CLR WR[1]                           ;EXPECTED DATA
U 177D, 9B9C,F5 ;18924          MEM.REQ[WRITE.V.WCHK] ADRS[#0] DT[LONG]
                ;18925                                              ; SET UP TO WRITE AT VIRTUAL 0 (PHY 200)
U 177E, B29C,15 ;18926          WRITE.MEM LS[ZERO]                  ;WRITE 0'S AT PHY 200
U 177F, 9853,F5 ;18927          MEM.REQ[READ.P] ADRS[#200] DT[LONG]
                ;18928                                              ;SET UP TO READ ADDRESS 200
U 1780, 3022,15 ;18929          MOV MEM.DATA TO WR[0]               ;GET RESULT
U 1781, 0869,3C ;18930          JSR [CHECK.RESULT]                  ;CHECK FOR DATA OF 0
U 1782, 0977,C4 ;18931          JMP [LOOP.T2F.1]                    ;ERROR LOOP IF ENABLED
                ;18932
U 1783, FF82,15 ;18933          INC LS[ERROR.NUMBER]                ;ERROR 2
                ;18934  LOOP.T2F.2:
U 1784, DF28,95 ;18935          MCOM LS[#FFFF] TO WR[1]             ;EXPECTED DATA (FFFF0000)
U 1785, 9B42,F5 ;18936          MEM.REQ[WRITE.V.WCHK] ADRS[#2] DT[LONG]
                ;18937                                              ; SET UP TO WRITE AT VIRTUAL 2 (PHY 202)
U 1786, 329E,15 ;18938          WRITE.MEM LS[ONES]                  ;WRITE 1'S AT PHY 202
U 1787, 9853,F5 ;18939          MEM.REQ[READ.P] ADRS[#200] DT[LONG]
                ;18940                                              ;SET UP TO READ ADDRESS 200
U 1788, 3022,15 ;18941          MOV MEM.DATA TO WR[0]               ;GET RESULT
U 1789, 0869,3C ;18942          JSR [CHECK.RESULT]                  ;CHECK FOR DATA OF 0000FFFF
U 178A, 8978,44 ;18943          JMP [LOOP.T2F.2]                    ;ERROR LOOP IF ENABLED
                ;18944
U 178B, FF82,15 ;18945          INC LS[ERROR.NUMBER]                ;ERROR 3
U 178C, B676,15 ;18946          MOV LS[MME] TO WR[0]                ;SET MME BIT IN WR
U 178D, C77A,15 ;18947          BIS LS[TB.PAR.DIAG] TO WR[0]        ;ALSO SET TB PAR ERR DIAG BIT
U 178E, 8A17,FC ;18948          JSR [WRITE.CSR1]                    ;WRITE INTO CSR 1
                ;18949  LOOP.T2F.3:
U 178F, DF28,95 ;18950          MCOM LS[#FFFF] TO WR[1]             ;EXPECTED DATA
U 1790, 9B9C,F5 ;18951          MEM.REQ[WRITE.V.WCHK] ADRS[#0] DT[LONG]
                ;18952                                              ; SET UP TO WRITE AT VIRTUAL 0 (PHY 200)
U 1791, B29C,15 ;18953          WRITE.MEM LS[ZERO]                  ;TRY TO WRITE 0'S AT PHY 200
U 1792, 9853,F5 ;18954          MEM.REQ[READ.P] ADRS[#200] DT[LONG]
                ;18955                                              ;SET UP TO READ ADDRESS 200
U 1793, 3022,15 ;18956          MOV MEM.DATA TO WR[0]               ;GET RESULT
U 1794, 0869,3C ;18957          JSR [CHECK.RESULT]                  ;CHECK FOR DATA OF 0000FFFF (NO CHANGE)
U 1795, 0978,F4 ;18958          JMP [LOOP.T2F.3]                    ;ERROR LOOP IF ENABLED
                ;18959
                ;18960  END.T2F:
```

```
;18961  .page   "  TEST 30 WRITE ARRAY Flow with ECC DIS or ERROR (M8391 MCT Module)"
;18962  ;++
;18963  ;
;18964  ; Test Description:
;18965  ;
;18966  ;       This test is designed to check for faults in the control store PROMs
;18967  ;       by verifying all paths of the micro-code.  This test checks the path
;18968  ;       taken when a WRITE.P is issued with ECC DIS set.  The second part of
;18969  ;       this test checks the micro path when a single or double bit error
;18970  ;       occurs during the read portion of a write.
;18971  ;         The micro code will write the CKBTs it generates for the write data
;18972  ;       into CSR 0 if ECC DIS is set before the write.  This is the only dif-
;18973  ;       ference from a write with ECC DIS clear.  This is tested by writing
;18974  ;       data of all 0's and data of a 1 in memory with ECC DIS set and checking
;18975  ;       CSR 0 for the correct CKBTs and the memory location for the correct
;18976  ;       data.  This test is done twice, once with the WRITE.MEM immediately
;18977  ;       following the MEM.REQ and once with 5 NOPs between the 2 parts of the
;18978  ;       write.  This will allow testing two different paths.
;18979  ;         The second part of the test checks the path when a data error occurs
;18980  ;       on an unaligned longword write (an aligned longword write writes all
;18981  ;       new CKBTs so does not care what the old ones were).
;18982  ;
;18983  ; Assumptions:
;18984  ;
;18985  ;       It is assumed that all previous tests have run successfully, and
;18986  ;       that all hardware (other than MCT control store PROMs) is operating
;18987  ;       correctly.
;18988  ;
;18989  ; Test Steps:
;18990  ;
;18991  ;       1)  Set up error mask, error number, and module number in LS (for
;18992  ;           error printouts) and clear previous error number in LS.
;18993  ;       2)  Set ECC DIS in CSR 1.  Write data of all 0's in memory address
;18994  ;           0.  Read CSR 0 and check for CKBTs for data of 0 (0111100).
;18995  ;       3)  Read address 0 and check for all 0's.
;18996  ;       4)  Write data of a 1 in memory address 0.  Read CSR 0 and check for
;18997  ;           CKBTs for data of a 1 (1100100).
;18998  ;       5)  Read address 0 and check for data of a 1.
;18999  ;       6)  Repeat steps 2 through 5 with a delay of 5 NOPs between the MEM.REQ
;19000  ;           and the WRITE.MEM when writing the data.  This checks a different
;19001  ;           path through the micro-code since CPU DR is delayed.
;19002  ;       7)  Write a 1 in memory location 0 with CKBTs for data of 0 using the
;19003  ;           MAINT.ECC.DATA routine.  Write a 10000 at address 4 with CKBTs for
;19004  ;           data of all 0's.  This puts a single error at both longword loc-
;19005  ;           ations.
;19006  ;       8)  Write an unaligned longword of all 1's at location 2.
;19007  ;       9)  Read location 0 and check for data of FFFF0000.
;19008  ;       10) Read location 4 and check for data of 0000FFFF.
;19009  ;       11) Repeat steps 7 through 10 with a delay of 5 NOPs between the MEM.REQ
;19010  ;           and the WRITE.MEM when writing the data.  This checks a different
;19011  ;           path through the micro-code since CPU DR is delayed.
;19012  ;       12) Write a 3 in memory location 0 with CKBTs for data of 0 using the
;19013  ;           MAINT.ECC.DATA routine.  Write all 3's at address 4 with normal
;19014  ;           CKBTs.  This puts a double bit error at address 0.
;19015  ;       13) Write an unaligned longword of all 1's at location 2.
```

```
;19016  ;       14) Read location 0 and check for data of 3 (no change since write is
;19017  ;           aborted).
;19018  ;       15) Read location 4 and check for data of all 0's.
;19019  ;       16) Repeat steps 12 through 14 with a delay of 5 NOPs between the MEM.REQ
;19020  ;           and the WRITE.MEM when writing the data.  This checks a different
;19021  ;           path through the micro-code since CPU DR is delayed.  Only the
;19022  ;           first location is checked this time since the different path is
;19023  ;           only on the first cycle of a 2 cycle write.
;19024  ;       17) Write a 3 in memory location 4 with CKBTs for data of 0 using the
;19025  ;           MAINT.ECC.DATA routine.  Write all 0's at address 0 with normal
;19026  ;           CKBTs.  This puts a double bit error at address 4.
;19027  ;       18) Write an unaligned longword of all 1's at location 2.
;19028  ;       19) Read location 0 and check for data of FFFF0000 (first cycle will
;19029  ;           complete since error occurs on second cycle).
;19030  ;       20) Read location 4 and check for data of 3 (write aborted).
;19031  ;
;19032  ; Errors:
;19033  ;
;19034  ;       NOTE:  Expected and received data is array contents when error mask is
;19035  ;              0,  CSR 0 data when mask is FFFFFF80.
;19036  ;
;19037  ;       Error 1 - WRITE.P with ECC DIS set failed to load CKBTs in CSR 0 cor-
;19038  ;                 rectly (CKBTs of 0111100).
;19039  ;       Error 2 - WRITE.P with ECC DIS set failed to write memory correctly
;19040  ;                 with data of all 0's.
;19041  ;       Error 3 - WRITE.P with ECC DIS set failed to load CKBTs in CSR 0 cor-
;19042  ;                 rectly (CKBTs of 1100100).
;19043  ;       Error 4 - WRITE.P with ECC DIS set failed to write memory correctly
;19044  ;                 with data of a 1.
;19045  ;       Errors 5 thru 8 - same as errors 1 thru 4 except a different path is
;19046  ;                 taken
;19047  ;       Error 9 - WRITE.P of unaligned longword failed to write address 0
;19048  ;                 correctly when address 0 contained a single bit error.
;19049  ;       Error A - WRITE.P of unaligned longword failed to write address 4
;19050  ;                 correctly when addresses 0 and 4 contained a single bit error.
;19051  ;       Errors B and C - Same as errors 9 and A above except with CPU DR delayed.
;19052  ;       Error D - WRITE.P of unaligned longword failed to abort first cycle
;19053  ;                 write on a double bit error.
;19054  ;       Error E - WRITE.P of unaligned longword failed to abort second cycle
;19055  ;                 write when a double bit error occurred on the first cycle.
;19056  ;       Error F - Same as error D above except with delayed CPU DR.
;19057  ;       Error 10 -WRITE.P of unaligned longword failed to write first cycle
;19058  ;                 correctly when a double error existed in the 2nd cycle.
;19059  ;       Error 11 -WRITE.P of unaligned longword failed to abort write on
;19060  ;                 second cycle when a double error existed in the 2nd cycle.
;19061  ;
;19062  ; Debug:
;19063  ;
;19064  ;       Note:  The following errors most likely indicate a problem with the
;19065  ;       micro-code flow.  The hardware used by the micro-code has been pre-
;19066  ;       viously tested.  To trace a problem, check the sequence outlined
;19067  ;       below.  If the sequence is OK, suspect a bad bit in one of the control
;19068  ;       store PROMs and check each state for the correct enables.
;19069  ;
;19070  ;       Error 1 - This error indicates a problem with the micro flow as-
```

```
;19071  ;       sociated with WRITE.P with ECC DIS set.  The flow up to WR ARY ALIGN
;19072  ;       LW C8 has been used previously.  From here the next word should branch
;19073  ;       on ECC DIS the word which asserts CSR CB/SYN CLK.  From there the flow
;19074  ;       should go to WR ARY ALIGN LW C11.
;19075  ;
;19076  ;       Error 2 - Same as error 1.  Suspect something in the cycle that asserts
;19077  ;       CSR CB/SYN CLK is fouling things up.
;19078  ;
;19079  ;       Error 3 - Same as error 1 with different data.
;19080  ;
;19081  ;       Error 4 - Same as error 2 with different data.
;19082  ;
;19083  ;       Error 5 - This flow takes a different path than the above errors be-
;19084  ;       cause CPU DR is not asserted at WR ARY ALIGN LW C7.  The flow up to WR
;19085  ;       ARY C10 has been used previously.  From here the ECC DIS branch to WR
;19086  ;       ARY C11 ECC DIS should be taken.  Then the flow should go to WR ARY
;19087  ;       C12.
;19088  ;
;19089  ;       Error 6 - Same as error 5.  Suspect something in WR ARY C11 ECC DIS
;19090  ;       is fouling things up.
;19091  ;
;19092  ;       Error 7 - Same as error 5 with different data.
;19093  ;
;19094  ;       Error 8 - Same as error 6 with different data.
;19095  ;
;19096  ;       Error 9 - This flow should take the ERR and CPU DR path out of WR ARY
;19097  ;       C7.  The word it goes to should take the SER path to WR ARY WSDE C8A
;19098  ;       to WR ARY WSDE C9 to WR ARY C8.  The rest of the first cycle write
;19099  ;       has been checked.
;19100  ;
;19101  ;       Error A - This flow should take the ERR path out of WR ARY C20 to WR
;19102  ;       ARY WDE C21 to WR AEY WSDE C22 to WR ARY WSDE C23 to WR ARY C21.  The
;19103  ;       rest of the 2nd cycle write has been checked previously.
;19104  ;
;19105  ;       Error B - Same as error 9 except the branch taken is ERR and CPU DR
;19106  ;       NOT to WR ARY WDE C8 to WR ARY WSDE C8A.  From there is same as error
;19107  ;       9.
;19108  ;
;19109  ;       Error C - Same as error A.
;19110  ;
;19111  ;       Error D - This flow should take the ERR and CPU DR path from WR ARY
;19112  ;       C7.  The following word should take the UNC ERR path to ABORT WR.WUDE
;19113  ;       to ABORT WR.WUDE C1 to ABORT WR.WUDE C2 to ABORT WR WUDE C3 to OCTA
;19114  ;       WRITE EXIT to IDLE.
;19115  ;
;19116  ;       Error E - Same as error D.  Should never have gotten to second cycle
;19117  ;       write.
;19118  ;
;19119  ;       Error F - Same as error D except should take ERR and CPU DR NOY path
;19120  ;       to WR ARY WDE C8 to ABORT WR.WUDE.
;19121  ;
;19122  ;       Error 10 - This flow should take the ERR NOT and CPU DR path from WR
;19123  ;       ARY C7 and continue as normal.
;19124  ;
;19125  ;       Error 11 - This path should take the SER NOT path from WR ARY WDE C21
```

```
                    ;19126  ;       to the insruction which goes to ABORT WR WUDE C1.  The rest is the
                    ;19127  ;       same as error D after ABORT WR WUDE C1.
                    ;19128  ;
                    ;19129  ;--
                    ;19130
                    ;19131  T.30:
U 1796, B65E,15     ;19132          MOV LS[BEGIN.TEST] TO WR[0]      ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;19133                                           ; STATUS WORD
U 1797, 3E80,15     ;19134          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;19135                                           ; BIT 15 INDICATES START OF TEST TO
                    ;19136                                           ; CONSOLE
U 1798, 10E0,15     ;19137          MISC [SET.CP.ATTN]               ;ISSUE CPU ATTN TO CONSOLE
                    ;19138  WAIT.T30.0:
U 1799, 8979,94     ;19139          JMP [WAIT.T30.0]                 ;LOOP HERE WAITING FOR CONSOLE TO
                    ;19140                                           ; RESPOND
                    ;19141
U 179A, 0A1A,AC     ;19142          JSR [SETUP.1]                    ;SET UP MASKS, MODULE CODE ETC.
                    ;19143
U 1798, 3713,15     ;19144          MOV LS[#FFFFFF80] TO WR[2]       ;BITS 7-31 SET. ERROR MASK TO CHECK
                    ;19145                                           ; BITS 0-6
U 179C, B641,95     ;19146          MOV LS[#1] TO WR[3]              ;1 IN WR FOR ERROR NUMBER
U 179D, 3672,15     ;19147          MOV LS[ECC.DIS] TO WR[0]         ;SET ECC DIS IN WR
U 179E, 8A17,FC     ;19148          JSR [WRITE.CSR1]                 ;SET ECC DIS IN CSR 1
                    ;19149
                    ;19150  LOOP.T30.2:
U 179F, BE83,95     ;19151          MOV WR[3] TO LS[ERROR.NUMBER]    ;SET UP ERROR NUMBER IN LS
U 17A0, BE88,15     ;19152          MOV WR[2] TO LS[ERROR.MASK]      ;CHECK BITS 6-0 ONLY
                    ;19153  LOOP.T30.1:
U 17A1, 3700,95     ;19154          MOV LS[CKBTS.0111100] TO WR[1]   ;COMP OF EXPECTED DATA
U 17A2, A0C2,95     ;19155          COM WR[1]                        ;EXPECTED CKBTS
U 17A3, 999C,75     ;19156          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                    ;19157                                           ;SET UP TO WRITE MEMORY ADDRESS 0
U 17A4, B29C,15     ;19158          WRITE.MEM LS[ZERO]               ;WRITE ALL 0'S IN ADDRESS 0
U 17A5, 9D9D,75     ;19159          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
U 17A6, 3022,15     ;19160          MOV MEM.DATA TO WR[0]            ;GET CSR0 CONTENTS
U 17A7, 0869,3C     ;1916'          JSR [CHECK.RESULT]               ;CHECK FOR CORRECT CKBTS
U 17A8, 097A,14     ;19162          JMP [LOOP.T30.1]                 ;ERROR LOOP IF ENABLED
                    ;19163
U 17A9, FF82,15     ;19164          INC LS[ERROR.NUMBER]             ;ERROR 2
U 17AA, 2F82,95     ;19165          CLR WR[1]                        ;EXPECTED DATA
U 17AB, E58A,15     ;19166          CLR LS[ERROR.MASK]               ;CHECK ALL BITS
U 17AC, 189D,F5     ;19167          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                    ;19168                                           ;SET UP TO READ ADDRESS 0
U 17AD, 3022,15     ;19169          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 17AE, 0869,3C     ;19170          JSR [CHECK.RESULT]               ;CHECK FOR DATA OF 0
U 17AF, 8979,F4     ;19171          JMP [LOOP.T30.2]                 ;ERROR LOOP IF ENABLED
                    ;19172
U 17B0, FF82,15     ;19173          INC LS[ERROR.NUMBER]             ;ERROR 3
U 17B1, 3683,95     ;19174          MOV LS[ERROR.NUMBER] TO WR[3]    ;STORE ERROR NUMBER IN WR[3]
                    ;19175  LOOP.T30.4:
U 17B2, BE83,95     ;19176          MOV WR[3] TO LS[ERROR.NUMBER]    ;SET UP ERROR NUMBER IN LS
U 17B3, BE88,15     ;19177          MOV WR[2] TO LS[ERROR.MASK]      ;CHECK BITS 6-0 ONLY
                    ;19178  LOOP.T30.3:
U 17B4, B704,95     ;19179          MOV LS[CKBTS.1100100] TO WR[1]   ;COMP OF EXPECTED DATA
U 17B5, A0C2,95     ;19180          COM WR[1]                        ;EXPECTED CKBTS
```

```
U 17B6, 999C,75 ;19181          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;19182                                  ;SET UP TO WRITE MEMORY ADDRESS 0
U 17B7, 3240,15 ;19183          WRITE.MEM LS[#1]        ;WRITE A 1 IN ADDRESS 0
U 17B8, 9D9D,75 ;19184          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
U 17B9, 3022,15 ;19185          MOV MEM.DATA TO WR[0]   ;GET CSR0 CONTENTS
U 17BA, 0869,3C ;19186          JSR [CHECK.RESULT]      ;CHECK FOR CORRECT CKBTS
U 17BB, 897B,44 ;19187          JMP [LOOP.T30.3]        ;ERROR LOOP IF ENABLED
                ;19188
U 17BC, FF82,15 ;19189          INC LS[ERROR.NUMBER]    ;ERROR 4
U 17BD, 3640,95 ;19190          MOV LS[#1] TO WR[1]     ;EXPECTED DATA
U 17BE, E58A,15 ;19191          CLR LS[ERROR.MASK]      ;CHECK ALL BITS
U 17BF, 189D,F5 ;19192          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;19193                                  ;SET UP TO READ ADDRESS 0
U 17C0, 3022,15 ;19194          MOV MEM.DATA TO WR[0]   ;GET RESULT
U 17C1, 0869,3C ;19195          JSR [CHECK.RESULT]      ;CHECK FOR DATA OF A 1
U 17C2, 897B,24 ;19196          JMP [LOOP.T30.4]        ;ERROR LOOP IF ENABLED
                ;19197
U 17C3, FF82,15 ;19198          INC LS[ERROR.NUMBER]    ;ERROR 5
U 17C4, 3683,95 ;19199          MOV LS[ERROR.NUMBER] TO WR[3]   ;STORE ERROR NUMBER IN WR[3]
                ;19200  LOOP.T30.6:
U 17C5, BE83,95 ;19201          MOV WR[3] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
U 17C6, BE88,15 ;19202          MOV WR[2] TO LS[ERROR.MASK]     ;CHECK BITS 6-0 ONLY
                ;19203  LOOP.T30.5:
U 17C7, 3700,95 ;19204          MOV LS[CKBTS.0111100] TO WR[1]  ;COMP OF EXPECTED DATA
U 17C8, A0C2,95 ;19205          COM WR[1]               ;EXPECTED CKBTS
U 17C9, 999C,75 ;19206          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;19207                                  ;SET UP TO WRITE MEMORY ADDRESS 0
U 17CA, 0A1B,4C ;19208          JSR [NOP.DELAY]         ;EXECUTE 5 CYCLE DELAY TO DELAY CPU DR
                ;19209                                  ; FROM BEING ASSERTED
U 17CB, B29C,15 ;19210          WRITE.MEM LS[ZERO]      ;WRITE ALL 0'S IN ADDRESS 0
U 17CC, 9D9D,75 ;19211          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
U 17CD, 3022,15 ;19212          MOV MEM.DATA TO WR[0]   ;GET CSR0 CONTENTS
U 17CE, 0869,3C ;19213          JSR [CHECK.RESULT]      ;CHECK FOR CORRECT CKBTS
U 17CF, 097C,74 ;19214          JMP [LOOP.T30.5]        ;ERROR LOOP IF ENABLED
                ;19215
U 17D0, FF82,15 ;19216          INC LS[ERROR.NUMBER]    ;ERROR 6
U 17D1, 2F82,95 ;19217          CLR WR[1]               ;EXPECTED DATA
U 17D2, E58A,15 ;19218          CLR LS[ERROR.MASK]      ;CHECK ALL BITS
U 17D3, 189D,F5 ;19219          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;19220                                  ;SET UP TO READ ADDRESS 0
U 17D4, 3022,15 ;19221          MOV MEM.DATA TO WR[0]   ;GET RESULT
U 17D5, 0869,3C ;19222          JSR [CHECK.RESULT]      ;CHECK FOR DATA OF 0
U 17D6, 897C,54 ;19223          JMP [LOOP.T30.6]        ;ERROR LOOP IF ENABLED
                ;19224
U 17D7, FF82,15 ;19225          INC LS[ERROR.NUMBER]    ;ERROR 7
U 17D8, 3683,95 ;19226          MOV LS[ERROR.NUMBER] TO WR[3]   ;STORE ERROR NUMBER IN WR[3]
                ;19227  LOOP.T30.8:
U 17D9, BE83,95 ;19228          MOV WR[3] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
U 17DA, BE88,15 ;19229          MOV WR[2] TO LS[ERROR.MASK]     ;CHECK BITS 6-0 ONLY
                ;19230  LOOP.T30.7:
U 17DB, B704,95 ;19231          MOV LS[CKBTS.1100100] TO WR[1]  ;COMP OF EXPECTED DATA
U 17DC, A0C2,95 ;19232          COM WR[1]               ;EXPECTED CKBTS
U 17DD, 999C,75 ;19233          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;19234                                  ;SET UP TO WRITE MEMORY ADDRESS 0
U 17DE, 0A1B,4C ;19235          JSR [NOP.DELAY]         ;EXECUTE 5 CYCLE DELAY TO DELAY ASSERT-
```

```
                ;19236                                          ; ING CPU DR
U 17DF, 3240,15 ;19237          WRITE.MEM LS[#1]                ;WRITE A 1 IN ADDRESS 0
U 17E0, 9D9D,75 ;19238          MEM.REQ[READ.CSR] ADRS[CSR0] DT[LONG]
U 17E1, 3022,15 ;19239          MOV MEM.DATA TO WR[0]           ;GET CSR0 CONTENTS
U 17E2, 0869,3C ;19240          JSR [CHECK.RESULT]              ;CHECK FOR CORRECT CKBTS
U 17E3, 897D,B4 ;19241          JMP [LOOP.T30.7]                ;ERROR LOOP IF ENABLED
                ;19242
U 17E4, FF82,15 ;19243          INC LS[ERROR.NUMBER]            ;ERROR 8
U 17E5, 3640,95 ;19244          MOV LS[#1] TO WR[1]             ;EXPECTED DATA
U 17E6, E58A,15 ;19245          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
U 17E7, 189D,F5 ;19246          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;19247                                          ;SET UP TO READ ADDRESS 0
U 17E8, 3022,15 ;19248          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 17E9, 0869,3C ;19249          JSR [CHECK.RESULT]              ;CHECK FOR DATA OF A 1
U 17EA, 097D,94 ;19250          JMP [LOOP.T30.8]                ;ERROR LOOP IF ENABLED
                ;19251
U 17EB, B700,15 ;19252          MOV LS[CKBTS.0111100] TO WR[0]  ;GET CKBTS FOR DATA OF 0 IN WR 0
U 17EC, A0C0,15 ;19253          COM WR[0]                       ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 17ED, 452C,15 ;19254          BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE
U 17EE, 8A17,FC ;19255          JSR [WRITE.CSR1]                ;PUT CKBTS FOR DATA OF 0 IN CSR 1, CLEAR
                ;19256                                          ; ALL OTHER BITS IN CSR
U 17EF, 1D40,F5 ;19257          MEM.REQ[MAINT.ECC.DATA] ADRS[#1] DT[LONG]
                ;19258                                          ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;19259                                          ; SET FOR BRANCH
U 17F0, 3240,15 ;19260          WRITE.MEM LS[#1]                ;WRITE A 1 AT LOCATION 0 WITH CKBTS FOR
                ;19261                                          ; DATA OF ALL 0'S
U 17F1, 3644,15 ;19262          MOV LS[#4] TO WR[0]             ;4 IN WR
U 17F2, 2040,15 ;19263          INC WR[0]                       ;5 IN WR
U 17F3, BE12,15 ;19264          MOV WR[0] TO LS[T9]             ;ADDRESS IN LS (4 PLUS BIT 0 SET)
U 17F4, 9D12,F5 ;19265          MEM.REQ[MAINT.ECC.DATA] ADRS[T9] DT[LONG]
                ;19266                                          ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;19267                                          ; SET FOR BRANCH
U 17F5, B260,15 ;19268          WRITE.MEM LS[#10000]            ;WRITE A 10000 AT LOCATION 4 WITH CKBTS
                ;19269                                          ; FOR DATA OF ALL 0'S
                ;19270
U 17F6, FF82,15 ;19271          INC LS[ERROR.NUMBER]            ;ERROR 9
U 17F7, 3683,95 ;19272          MOV LS[ERROR.NUMBER] TO WR[3]   ;SAVE ERROR IN WR
U 17F8, E58A,15 ;19273          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
                ;19274  LOOP.T30.A:
U 17F9, BE83,95 ;19275          MOV WR[3] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER
                ;19276  LOOP.T30.9:
U 17FA, DF28,95 ;19277          MCOM LS[#FFFF] TO WR[1]         ;EXPECTED DATA (FFFF0000(H))
U 17FB, 9942,75 ;19278          MEM.REQ[WRITE.P] ADRS[#2] DT[LONG]
                ;19279                                          ;SET UP TO WRITE UNALIGNED LONGWORD
U 17FC, 329E,15 ;19280          WRITE.MEM LS[ONES]              ;WRITE ONES AT UNALIGNED LONGWORD
U 17FD, 189D,F5 ;19281          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;19282                                          ;SET UP TO READ ADDRESS 0
U 17FE, 3022,15 ;19283          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 17FF, 0869,3C ;19284          JSR [CHECK.RESULT]              ;CHECK FOR ERROR
U 1800, 897F,A4 ;19285          JMP [LOOP.T30.9]                ;ERROR LOOP IF ENABLED
                ;19286
U 1801, FF82,15 ;19287          INC LS[ERROR.NUMBER]            ;ERROR A
U 1802, B628,95 ;19288          MOV LS[#FFFF] TO WR[1]          ;EXPECTED DATA (0000FFFF(H))
U 1803, 1845,F5 ;19289          MEM.REQ[READ.P] ADRS[#4] DT[LONG]
                ;19290                                          ;SET UP TO READ 2ND LONGWORD
```

```
U 1804, 3022,15 ;19291          MOV MEM.DATA TO WR[0]              ;READ ADDRESS 4
U 1805, 0869,3C ;19292          JSR [CHECK.RESULT]                ;CHECK FOR CORRECT DATA
U 1806, 897F,94 ;19293          JMP [LOOP.T30.A]                  ;ERROR LOOP IF ENABLED
                ;19294
U 1807, B700,15 ;19295          MOV LS[CKBTS.0111100] TO WR[0]    ;GET CKBTS FOR DATA OF 0 IN WR 0
U 1808, A0C0,15 ;19296          COM WR[0]                         ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 1809, 452C,15 ;19297          BIC LS[#FFFFFF00] TO WR[0]        ;CLEAR ALL BUT LOW BYTE
U 180A, 8A17,FC ;19298          JSR [WRITE.CSR1]                  ;PUT CKBTS FOR DATA OF 0 IN CSR 1, CLEAR
                ;19299                                            ; ALL OTHER BITS IN CSR
U 180B, 1D40,F5 ;19300          MEM.REQ[MAINT.ECC.DATA] ADRS[#1] DT[LONG]
                ;19301                                            ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;19302                                            ; SET FOR BRANCH
U 180C, 3240,15 ;19303          WRITE.MEM LS[#1]                  ;WRITE A 1 AT LOCATION 0 WITH CKBTS FOR
                ;19304                                            ; DATA OF ALL 0'S
U 180D, 9D12,F5 ;19305          MEM.REQ[MAINT.ECC.DATA] ADRS[T9] DT[LONG]
                ;19306                                            ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;19307                                            ; SET FOR BRANCH
U 180E, B260,15 ;19308          WRITE.MEM LS[#10000]              ;WRITE A 10000 AT LOCATION 4 WITH CKBTS
                ;19309                                            ; FOR DATA OF ALL 0'S
                ;19310
U 180F, FF82,15 ;19311          INC LS[ERROR.NUMBER]              ;ERROR B
U 1810, 3683,95 ;19312          MOV LS[ERROR.NUMBER] TO WR[3]     ;SAVE ERROR IN WR
U 1811, 9944,75 ;19313          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                ;19314                                            ;SET UP TO WRITE 2ND LONGWORD
U 1812, B29C,15 ;19315          WRITE.MEM LS[ZERO]                ;WRITE ZEROS AT ADDRESS 4
U 1813, E58A,15 ;19316          CLR LS[ERROR.MASK]                ;CHECK ALL BITS
                ;19317  LOOP.T30.C:
U 1814, BE83,95 ;19318          MOV WR[3] TO LS[ERROR.NUMBER]     ;SET UP ERROR NUMBER
                ;19319  LOOP.T30.B:
U 1815, DF28,95 ;19320          MCOM LS[#FFFF] TO WR[1]           ;EXPECTED DATA (FFFF0000(H))
U 1816, 9942,75 ;19321          MEM.REQ[WRITE.P] ADRS[#2] DT[LONG]
                ;19322                                            ;SET UP TO WRITE UNALIGNED LONGWORD
U 1817, 0A1B,4C ;19323          JSR [NOP.DELAY]                   ;EXECUTE 5 CYCLE DELAY TO DELAY CPUDR
U 1818, 329E,15 ;19324          WRITE.MEM LS[ONES]                ;WRITE ONES AT UNALIGNED LONGWORD
U 1819, 189D,F5 ;19325          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;19326                                            ;SET UP TO READ ADDRESS 0
U 181A, 3022,15 ;19327          MOV MEM.DATA TO WR[0]             ;GET RESULT
U 181B, 0869,3C ;19328          JSR [CHECK.RESULT]                ;CHECK FOR ERROR
U 181C, 0981,54 ;19329          JMP [LOOP.T30.B]                  ;ERROR LOOP IF ENABLED
                ;19330
U 181D, FF82,15 ;19331          INC LS[ERROR.NUMBER]              ;ERROR C
U 181E, B628,95 ;19332          MOV LS[#FFFF] TO WR[1]            ;EXPECTED DATA (0000FFFF(H))
U 181F, 1845,F5 ;19333          MEM.REQ[READ.P] ADRS[#4] DT[LONG]
                ;19334                                            ;SET UP TO READ 2ND LONGWORD
U 1820, 3022,15 ;19335          MOV MEM.DATA TO WR[0]             ;READ ADDRESS 4
U 1821, 0869,3C ;19336          JSR [CHECK.RESULT]                ;CHECK FOR CORRECT DATA
U 1822, 8981,44 ;19337          JMP [LOOP.T30.C]                  ;ERROR LOOP IF ENABLED
                ;19338
U 1823, FF82,15 ;19339          INC LS[ERROR.NUMBER]              ;ERROR D
U 1824, 3683,95 ;19340          MOV LS[ERROR.NUMBER] TO WR[3]     ;SAVE IN WR
U 1825, 1D40,F5 ;19341          MEM.REQ[MAINT.ECC.DATA] ADRS[#1] DT[LONG]
                ;19342                                            ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;19343                                            ; SET FOR BRANCH
U 1826, B2C0,15 ;19344          WRITE.MEM LS[#3(H)]               ;WRITE A 3 AT LOCATION 0 WITH CKBTS FOR
                ;19345                                            ; DATA OF ALL 0'S (DOUBLE ERROR)
```

```
U 1827, 9944,75 ;19346          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                ;19347                                  ;SET UP TO WRITE AT ADDRESS 4
U 1828, B29C,15 ;19348          WRITE.MEM LS[ZERO]      ;WRITE 0'S IN ADDRESS 4
                ;19349  LOOP.T30.E:
U 1829, BE83,95 ;19350          MOV WR[3] TO LS[ERROR.NUMBER]   ;RESTORE ERROR NUMBER
                ;19351  LOOP.T30.D:
U 182A, B6C0,95 ;19352          MOV LS[#3(H)] TO WR[1]  ;EXPECTED DATA (WRITE ABORTED)
U 182B, 9942,75 ;19353          MEM.REQ[WRITE.P] ADRS[#2] DT[LONG]
                ;19354                                  ;SET UP TO WRITE UNALIGNED DATA
U 182C, 329E,15 ;19355          WRITE.MEM LS[ONES]      ;TRY TO WRITE ONES (DOUBLE ERROR IN 0
                ;19356                                  ; SHOULD PREVENT WRITE)
U 182D, 189D,F5 ;19357          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;19358                                  ;SET UP TO READ ADDRESS 0
U 182E, 3022,15 ;19359          MOV MEM.DATA TO WR[0]   ;GET RESULT
U 182F, 0869,3C ;19360          JSR [CHECK.RESULT]      ;CHECK FOR ERRORS
U 1830, 0982,A4 ;19361          JMP [LOOP.T30.D]        ;ERROR LOOP IF ENABLED
                ;19362
U 1831, FF82,15 ;19363          INC LS[ERROR.NUMBER]    ;ERROR E
U 1832, 2F82,95 ;19364          CLR WR[1]               ;EXPECTED DATA
U 1833, 1845,F5 ;19365          MEM.REQ[READ.P] ADRS[#4] DT[LONG]
                ;19366                                  ;SET UP TO READ ADDRESS 4
U 1834, 3022,15 ;19367          MOV MEM.DATA TO WR[0]   ;GET RESULT
U 1835, 0869,3C ;19368          JSR [CHECK.RESULT]      ;CHECK FOR ERRORS
U 1836, 0982,94 ;19369          JMP [LOOP.T30.E]        ;ERROR LOOP IF ENABLED
                ;19370
U 1837, FF82,15 ;19371          INC LS[ERROR.NUMBER]    ;ERROR F
U 1838, 1D40,F5 ;19372          MEM.REQ[MAINT.ECC.DATA] ADRS[#1] DT[LONG]
                ;19373                                  ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;19374                                  ; SET FOR BRANCH
U 1839, B2C0,15 ;19375          WRITE.MEM LS[#3(H)]     ;WRITE A 3 AT LOCATION 0 WITH CKBTS FOR
                ;19376                                  ; DATA OF ALL 0'S (DOUBLE ERROR)
                ;19377  LOOP.T30.F:
U 183A, B6C0,95 ;19378          MOV LS[#3(H)] TO WR[1]  ;EXPECTED DATA (WRITE ABORTED)
U 183B, 9942,75 ;19379          MEM.REQ[WRITE.P] ADRS[#2] DT[LONG]
                ;19380                                  ;SET UP TO WRITE UNALIGNED DATA
U 183C, 0A1B,4C ;19381          JSR [NOP.DELAY]         ;EXECUTE 5 CYCLE DELAY TO DELAY CPUDR
U 183D, 329E,15 ;19382          WRITE.MEM LS[ONES]      ;TRY TO WRITE ONES (DOUBLE ERROR IN 0
                ;19383                                  ; SHOULD PREVENT WRITE)
U 183E, 189D,F5 ;19384          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;19385                                  ;SET UP TO READ ADDRESS 0
U 183F, 3022,15 ;19386          MOV MEM.DATA TO WR[0]   ;GET RESULT
U 1840, 0869,3C ;19387          JSR [CHECK.RESULT]      ;CHECK FOR ERRORS
U 1841, 8983,A4 ;19388          JMP [LOOP.T30.F]        ;ERROR LOOP IF ENABLED
                ;19389
U 1842, FF82,15 ;19390          INC LS[ERROR.NUMBER]    ;ERROR 10
U 1843, 3683,95 ;19391          MOV LS[ERROR.NUMBER] TO WR[3]   ;SAVE IN WR
U 1844, 9D12,F5 ;19392          MEM.REQ[MAINT.ECC.DATA] ADRS[T9] DT[LONG]
                ;19393                                  ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;19394                                  ; SET FOR BRANCH
U 1845, B2C0,15 ;19395          WRITE.MEM LS[#3(H)]     ;WRITE A 3 AT LOCATION 4 WITH CKBTS FOR
                ;19396                                  ; DATA OF ALL 0'S (DOUBLE ERROR)
U 1846, 999C,75 ;19397          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;19398                                  ;SET UP TO WRITE AT ADDRESS 0
U 1847, B29C,15 ;19399          WRITE.MEM LS[ZERO]      ;WRITE 0'S IN ADDRESS 0
                ;19400  LOOP.T30.11:
```

```
U 1848, BE83,95 ;19401          MOV WR[3] TO LS[ERROR.NUMBER]   ;RESTORE ERROR NUMBER
                ;19402  LOOP.T30.10:
U 1849, DF28,95 ;19403          MCOM LS[#FFFF] TO WR[1]         ;EXPECTED DATA (FIRST HALF OF 2 CYCLE
                ;19404                                          ; WRITE WILL COMPLETE)
U 184A, 9942,75 ;19405          MEM.REQ[WRITE.P] ADRS[#2] DT[LONG]
                ;19406                                          ;SET UP TO WRITE UNALIGNED DATA
U 184B, 329E,15 ;19407          WRITE.MEM LS[ONES]              ;WRITE ONES AT ADDRESS 2
U 184C, 189D,F5 ;19408          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;19409                                          ;SET UP TO READ ADDRESS 0
U 184D, 3022,15 ;19410          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 184E, 0869,3C ;19411          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 184F, 0984,94 ;19412          JMP [LOOP.T30.10]               ;ERROR LOOP IF ENABLED
                ;19413
U 1850, FF82,15 ;19414          INC LS[ERROR.NUMBER]            ;ERROR 11
U 1851, B6C0,95 ;19415          MOV LS[#3(H)] TO WR[1]          ;EXPECTED DATA (SECOND CYCLE SHOULD HAVE
                ;19416                                          ; ABORTED)
U 1852, 1845,F5 ;19417          MEM.REQ[READ.P] ADRS[#4] DT[LONG]
                ;19418                                          ;SET UP TO READ ADDRESS 4
U 1853, 3022,15 ;19419          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1854, 0869,3C ;19420          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 1855, 8984,84 ;19421          JMP [LOOP.T30.11]               ;ERROR LOOP IF ENABLED
                ;19422
                ;19423  END.T30:
```

```
;19424  .page   "  TEST 31 WRITE OCTA Flow  (M8391 MCT Module)"
;19425  ;++
;19426  ;
;19427  ; Test Description:
;19428  ;
;19429  ;       This test is designed to check for faults in the control store PROMs
;19430  ;       by verifying all paths of the micro-code.  This test checks the path
;19431  ;       taken when a OCTA.WRITE.P or OCTA.WRITE.V.WCHK is issued.  This test
;19432  ;       will also check the paths taken when a page boundary is crossed.
;19433  ;         The first part of the test simply writes a background of ones in
;19434  ;       the first 4 longwords of memory.  Then the OCTA.WRITE.P is executed
;19435  ;       with data patterns 1, 2, 4, and 8 respectively in these locations.  The
;19436  ;       data is checked with the READ.P micro flow.  This will check most of
;19437  ;       the OCTA.WRITE flow.  This test is repeated with a delay of 5 NOPs
;19438  ;       between the MEM.REQ and each WRITE.MEM to delay CPU DR.  This will test
;19439  ;       the branch path on CPU DR not.
;19440  ;         The next part checks the OCTA.WR.V.WCHK flow, which simply starts
;19441  ;       at the entry point one cycle after the OCTA.WRITE.P entry.  The write
;19442  ;       abort is also checked by forcing a TB parity error next.
;19443  ;         Finally the PG BOUND branch is checked as well as an error sum branch
;19444  ;       that occurs after a page boundary is crossed.
;19445  ;
;19446  ; Assumptions:
;19447  ;
;19448  ;       It is assumed that all previous tests have run successfully, and
;19449  ;       that all hardware (other than MCT control store PROMs) is operating
;19450  ;       correctly.
;19451  ;
;19452  ; Test Steps:
;19453  ;
;19454  ;       1)  Set up error mask, error number, and module number in LS (for
;19455  ;           error printouts) and clear previous error number in LS.
;19456  ;       2)  Write a background of ones in the first four memory longwords.
;19457  ;       3)  Write data of 1, 2, 4, and 8 in the first four longwords of
;19458  ;           memory using the OCTA.WRITE.P flow.
;19459  ;       4)  Verify that the four longwords were written correctly.
;19460  ;       5)  Repeat steps 2 through 4 with delays between the MEM.REQ and the
;19461  ;           WRITE.MEM instructions to force the CPU DR not path in the micro
;19462  ;           flows.
;19463  ;       6)  Set up TB to map virtual address 0 to physical address 200.  Set
;19464  ;           MME bit in CSR 1.  Write a background of ones at 4 longwords start-
;19465  ;           ing at address 200.
;19466  ;       7)  Execute OCTA.WR.V.WCHK at virtual address 0.  Check the result at
;19467  ;           physical address 200 through 20F.  This verifies octa virtual write
;19468  ;           dispatch.
;19469  ;       8)  Set MME and TB PAR DIAG bits in CSR 1.  Write a background of 1's
;19470  ;           at addresses 200-20F.
;19471  ;       9)  Execute OCTA.WR.V.WCHK at virtual address 0.  Verify that write is
;19472  ;           aborted due to TB parity error by checking that address 200-20F are
;19473  ;           not modified.
;19474  ;       10) Set MME in CSR 1.  Write TB address 1 to map virtual 0 to physical
;19475  ;           200 and clear VALID bit to force an error sum.  Write TB address 0
;19476  ;           to map virtual 0 to phsical 0 without an error.
;19477  ;       11) Write all 1's in four longwords starting at physical address 1FC.
;19478  ;       12) Execute OCTA.WR.V.WCHK starting a virtual 1FC.  The second write
```

```
;19479  ;           will cross a page boundary to test the PAGE BOUND branch and should
;19480  ;           abort due to ERROR SUM.
;19481  ;       13) Read address 1FC(H) and check for a 1(H).  Read remaining three
;19482  ;           longwords checking for no change (write aborted).
;19483  ;       14) Repeat steps 11 and 12 starting at address 1F8.  The third write
;19484  ;           will cross a page boundary.
;19485  ;       15) Check addresses 1F8 and 1FC for a 1 and a 2 respectively.  Check
;19486  ;           addresses 200 and 204 for all 1's (unmodified data).
;19487  ;       16) Set VALID bit in TB 1 to eliminate the error.
;19488  ;       17) Repeat steps 11 and 12.  This time no abort should take place
;19489  ;           and all 4 longwords should have the correct data in them.
;19490  ;       18) Repeat step 17 with a delay between the first and second longword
;19491  ;           write.  This checks the CPU DR not path in the page bound flow.
;19492  ;
;19493  ; Errors:
;19494  ;
;19495  ;       NOTE:  Data printed under OTHER in error report is address of failing
;19496  ;       location.
;19497  ;
;19498  ;       Error 1 - OCTA.WRITE.P failed to write 4 longwords correctly (CPU DR
;19499  ;                 was not delayed).
;19500  ;       Error 2 - OCTA.WRITE.P failed to write 4 longwords correctly (CPU DR
;19501  ;                 delayed).
;19502  ;       Error 3 - OCTA.WR.V.WCHK failed.  Suspect dispatch to entry point.
;19503  ;       Error 4 - OCTA.WR.V.WCHK failed to abort on errsum (TB parity error).
;19504  ;       Error 5 - OCTA.WR.V.WCHK failed to write first longword.
;19505  ;       Error 6 - OCTA.WR.V.WCHK failed to abort on PAGE BOUND path with TB
;19506  ;                 error (not valid error).
;19507  ;       Error 7 - OCTA.WR.V.WCHK failed to write first two longwords.
;19508  ;       Error 8 - OCTA.WR.V.WCHK failed to take PAGE BOUND path with TB error
;19509  ;                 (not valid error) on the third longword write to memory.
;19510  ;       Error 9 - OCTA.WR.V.WCH failed to write 4 longwords correctly when
;19511  ;                 a PG BOUNDARY was crossed, even though no ERROR SUM was
;19512  ;                 generated.
;19513  ;       Error A - OCTA.WR.V.WCH failed to write 4 longwords correctly when
;19514  ;                 a PG BOUNDARY was crossed, even though no ERROR SUM was
;19515  ;                 generated (delayed CPU DR).
;19516  ;
;19517  ; Debug:
;19518  ;
;19519  ;       Note:  The following errors most likely indicate a problem with the
;19520  ;       micro-code flow.  The hardware used by the micro-code has been pre-
;19521  ;       viously tested.  To trace a problem, check the sequence outlined
;19522  ;       below.  If the sequence is OK, suspect a bad bit in one of the control
;19523  ;       store PROMs and check each state for the correct enables.
;19524  ;
;19525  ;       Error 1 - This is the first time that an OCTA WRITE is tested, so any
;19526  ;       instruction in the micro flow could be causing a problem.  The path
;19527  ;       that should be observed is as follows:  WRITE PH.OCTA to OCTA WRITE
;19528  ;       V WCHK to OCTA WRITE C4 and continuing through OCTA WRITE C19 con-
;19529  ;       secutively.  OCTA WRITE C8 through OCTA WRITE C16 is executed next.
;19530  ;       Then the ICO branch is taken to OCTA WRITE C17A to OCTA WRITE EXIT to
;19531  ;       IDLE.  Note that the ERROR SUM not branch, the PG BOUND not branch, the
;19532  ;       L CPU DR branch and the P2 branch is taken throughout the micro flow.
;19533  ;
```

```
                    ;19534  ;       Error 2 - Same as error 1 above except the CPU DR not path will be
                    ;19535  ;       taken throughout the micro flow.
                    ;19536  ;
                    ;19537  ;       Error 3 - This error most likely indicates a problem on the dispatch.
                    ;19538  ;       The first address entered at dispatch should be OCTA WRITE V WCHK
                    ;19539  ;       instead of WRITE PH.OCTA.  The rest of the flow is identical to error
                    ;19540  ;       1.
                    ;19541  ;
                    ;19542  ;       Error 4 - This error indicates a problem with the ERRSUM branch at OCTA
                    ;19543  ;       WRITE C6 or the error sum clock in OCTA WRITE C5.  The flow is from
                    ;19544  ;       OCTA WRITE V WCHK to OCTA WRITE C4 to OCTA WRITE C5 to OCTA WRITE C6
                    ;19545  ;       through the ERRSUM path to ABORT WRITE CYC C1.
                    ;19546  ;
                    ;19547  ;       Error 5 - This error is unlikely.  It indicates a problem in the first
                    ;19548  ;       part of the OCTA.WR.V.WCHK flow that was tested previously.  The only
                    ;19549  ;       difference is that a PAGE BOUNDARY will be crossed on the next write.
                    ;19550  ;       Run previous tests again if this fails.
                    ;19551  ;
                    ;19552  ;       Error 6 - This error indicates a problem with the PAGE BOUND branch at
                    ;19553  ;       OCTA WRITE C12 or the ERRSUM branch at the end of the PAGE BOUND path.
                    ;19554  ;       At OCTA WRITE C12 the micro code should take the PG BOUND path to OCTA
                    ;19555  ;       WRITE C13A to OCTA WRITE PG BND C14 to the cycle that issues the ERR
                    ;19556  ;       SUM clocks to the cycle that checks for ERRSUM and CPU DR.  The ERRSUM
                    ;19557  ;       path should be taken to the cycle that goes to ABORT WRITE CYC C1.
                    ;19558  ;
                    ;19559  ;       Error 7 - Same as error 5
                    ;19560  ;
                    ;19561  ;       Error 8 - This error indicates a problem with the PAGE BOUND branch at
                    ;19562  ;       OCTA WRITE C18.  The micro code should take the PG BOUND path at OCTA
                    ;19563  ;       WRITE C18 and go to the cycle that goes to OCTA WRITE C4.  The rest of
                    ;19564  ;       the path has been tested previously.
                    ;19565  ;
                    ;19566  ;       Error 9 - This error indicates a problem with the ERRSUM not branch in
                    ;19567  ;       the PG BOUND path at OCTA WRITE C12.  At the end of this path, the
                    ;19568  ;       micro word which branches on ERRSUM and CPU DR should take the CPU DR
                    ;19569  ;       and ERRSUM not path to OCTA WRITE C14.
                    ;19570  ;
                    ;19571  ;       Error A - This error indicates a problem with the CPU DR branch in the
                    ;19572  ;       PG BOUND path at OCTA WRITE C12.    At the end of this path, the
                    ;19573  ;       micro word which branches on ERRSUM and CPU DR should take the CPU DR
                    ;19574  ;       not and ERRSUM not path to OCTA WRITE WAIT CPUDR C2.
                    ;19575  ;
                    ;19576  ;--
                    ;19577
                    ;19578  T.31:
U 1856, B65E,15     ;19579          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;19580                                          ; STATUS WORD
U 1857, 3E80,15     ;19581          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;19582                                          ; BIT 15 INDICATES START OF TEST TO
                    ;19583                                          ; CONSOLE
U 1858, 10E0,15     ;19584          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;19585  WAIT.T31.0:
U 1859, 8985,94     ;19586          JMP [WAIT.T31.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;19587                                          ; RESPOND
                    ;19588
```

```
U 185A, 0A1A,9C ;19589          JSR [SETUP]                         ;SET UP MASKS, MODULE CODE ETC. AND
                ;19590                                              ; CLEAR MCT CSR 1
U 185B, B645,15 ;19591          MOV LS[#4] TO WR[2]                 ;INCREMENT VALUE FOR ADDRESS
U 185C, 3649,95 ;19592          MOV LS[#10] TO WR[3]                ;GET A 10(H) IN WR
U 185D, 0995,2C ;19593          JSR [BACK.T31]                      ;WRITE ALL 1'S IN FIRST 4 LONGWORDS
                ;19594
U 185E, B670,15 ;19595          MOV LS[OTHER.DATA] TO WR[0]         ;SET BIT TO PRINT UNDER OTHER IN ERROR
                ;19596                                              ; REPORT
U 185F, 3E80,15 ;19597          MOV WR[0] TO LS[CONTROL.STATUS]     ;PUT IN CONTROL AND STATUS WORD
                ;19598  LOOP.T31.1:
U 1860, 9C9C,75 ;19599          MEM.REQ[OCTA.WRITE.P] ADRS[#0] DT[LONG]
                ;19600                                              ;SET UP TO WRITE MEMORY ADDRESS 0-F(H)
U 1861, 3240,15 ;19601          WRITE.MEM LS[#1]                    ;WRITE A 1 AT LONGWORD ADDRESS 0
U 1862, B242,15 ;19602          WRITE.MEM LS[#2]                    ;WRITE A 2 AT LONGWORD ADDRESS 4
U 1863, B244,15 ;19603          WRITE.MEM LS[#4]                    ;WRITE A 4 AT LONGWORD ADDRESS 8
U 1864, 3246,15 ;19604          WRITE.MEM LS[#8]                    ;WRITE A 8 AT LONGWORD ADDRESS C
                ;19605
U 1865, 6588,15 ;19606          CLR LS[ADDRESS.DATA]                ;CLEAR ADDRESS FOR PRINTOUT
U 1866, E5F8,15 ;19607          CLR LS[OS]                          ;CLEAR INDEX
U 1867, 6512,15 ;19608          CLR LS[T9]                          ;START READING AT ADDRESS 0
                ;19609  READ.T31.1:
U 1868, 36F2,95 ;19610          MOV LS[SHIFT.OS(3-0)] TO WR[1]      ;GET EXPECTED DATA
U 1869, 1813,F5 ;19611          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;19612                                              ;SET UP TO READ MEMORY
U 186A, 3022,15 ;19613          MOV MEM.DATA TO WR[0]               ;GET RESULT
U 186B, 0869,3C ;19614          JSR [CHECK.RESULT]                  ;CHECK FOR CORRECT DATA
U 186C, 8986,04 ;19615          JMP [LOOP.T31.1]                    ;ERROR LOOP IF ENABLED
                ;19616
U 186D, 7FF8,15 ;19617          INC LS[OS]                          ;INCREMENT INDEX
U 186E, EC13,15 ;19618          ADD WR[2] TO LS[T9]                 ;ADD 4 TO MEMORY ADDRESS
U 186F, EC89,15 ;19619          ADD WR[2] TO LS[ADDRESS.DATA]       ;ADD 4 TO ADDRESS TO PRINT
                ;19620          CMP WR[3] WITH LS[T9],              ;SEE IF DONE (WR 3 CONTAINS 10(H))
U 1870, 4F13,B5 ;19621           DT(LONG)&SET.ALU.CC                ; SET CONDITION CODES
U 1871, 0986,81 ;19622          JMP.IF[NEQ] TO [READ.T31.1]         ;REPEAT UNTIL DONE
                ;19623
U 1872, FF82,15 ;19624          INC LS[ERROR.NUMBER]                ;ERROR 2
U 1873, 0995,2C ;19625          JSR [BACK.T31]                      ;WRITE ALL 1'S IN FIRST 4 LOCATIONS
                ;19626  LOOP.T31.2:
U 1874, 9C9C,75 ;19627          MEM.REQ[OCTA.WRITE.P] ADRS[#0] DT[LONG]
                ;19628                                              ;SET UP TO WRITE MEMORY ADDRESS 0-F(H)
U 1875, E5F8,15 ;19629          CLR LS[OS]                          ;CLEAR INDEX
U 1876, DB00,15 ;19630          NOP                                 ;DELAY CPU DR
U 1877, DB00,15 ;19631          NOP                                 ;DELAY CPU DR
                ;19632  WRITE.T31.2:
U 1878, DB00,15 ;19633          NOP                                 ;DELAY CPU DR
U 1879, DB00,15 ;19634          NOP                                 ;DELAY CPU DR
U 187A, 32F2,15 ;19635          WRITE.MEM LS[SHIFT.OS(3-0)]         ;WRITE DATA AT LONGWORD
U 187B, 7FF8,15 ;19636          INC LS[OS]                          ;INCREMENT INDEX
                ;19637          CMP WR[3] WITH LS[OS],              ;SEE IF DONE (WR 3 CONTAINS 10(H))
U 187C, CFF9,B5 ;19638           DT(LONG)&SET.ALU.CC                ; SET CONDITION CODES
U 187D, 8987,81 ;19639          JMP.IF[NEQ] TO [WRITE.T31.2]        ;REPEAT UNTIL DONE
                ;19640
U 187E, 6588,15 ;19641          CLR LS[ADDRESS.DATA]                ;CLEAR ADDRESS FOR PRINTOUT
U 187F, E5F8,15 ;19642          CLR LS[OS]                          ;CLEAR INDEX
U 1880, 6512,15 ;19643          CLR LS[T9]                          ;START READING AT ADDRESS 0
```

```
                 ;19644  READ.T31.2:
U 1881, 36F2,95  ;19645          MOV LS[SHIFT.OS(3-0)] TO WR[*]  ;GET EXPECTED DATA
U 1882, 1813,F5  ;19646          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                 ;19647                                          ;SET UP TO READ MEMORY
U 1883, 3022,15  ;19648          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1884, 0869,3C  ;19649          JSR [CHECK.RESULT]              ;CHECK FOR CORRECT DATA
U 1885, 8987,44  ;19650          JMP [LOOP.T31.2]                ;ERROR LOOP IF ENABLED
                 ;19651
U 1886, 7FF8,15  ;19652          INC LS[OS]                      ;INCREMENT INDEX
U 1887, EC13,15  ;19653          ADD WR[2] TO LS[T9]              ;ADD 4 TO MEMORY ADDRESS
U 1888, EC89,15  ;19654          ADD WR[2] TO LS[ADDRESS.DATA]    ;ADD 4 TO ADDRESS TO PRINT
                 ;19655          CMP WR[3] WITH LS[T9],          ;SEE IF DONE (WR 3 CONTAINS 10(H))
U 1889, 4F13,B5  ;19656           DI(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 188A, 8988,11  ;19657          JMP.IF[NEQ] TO [READ.T31.2]     ;REPEAT UNTIL DONE
                 ;19658
U 188B, FF82,15  ;19659          INC LS[ERROR.NUMBER]            ;ERROR 3
U 188C, 367E,15  ;19660          MOV LS[BIT31] TO WR[0]          ;SET BIT 31 IN WR
U 188D, C77A,15  ;19661          BIS LS[BIT29] TO WR[0]          ;AND BIT 29
U 188E, 4774,15  ;19662          BIS LS[BIT26] TO WR[0]          ;AND BIT 26
U 188F, 4772,15  ;19663          BIS LS[BIT25] TO WR[0]          ;AND BIT 25
U 1890, C740,15  ;19664          BIS LS[BIT0] TO WR[0]           ;AND BIT 0
U 1891, 3E10,15  ;19665          MOV WR[0] TO LS[T8]             ;PUT IN TEMP LS
U 1892, 989C,F5  ;19666          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                 ;19667                                          ;SET UP TO WRITE TB ADDRESS 0
U 1893, 3210,15  ;19668          WRITE.MEM LS[T8]                ;SET VALID,PROT C,MODIFY,AND BYTE OFFSET
                 ;19669                                          ; BITS AND MAP VIRTUAL ADDRESS 0 TO
                 ;19670                                          ; PHYSICAL ADDRESS 200 IN TB
U 1894, 0A17,DC  ;19671          JSR [WRITE.CSR1.MME]            ;WRITE IN CSR 1
U 1895, B653,95  ;19672          MOV LS[#200] TO WR[3]           ;200 IN WR
U 1896, BE13,95  ;19673          MOV WR[3] TO LS[T9]             ;STORE ADDRESS IN TEMP LS
U 1897, 4749,95  ;19674          BIS LS[#10] TO WR[3]            ;WR[3]=210(H) (ADDRESS WHEN WRITE DONE)
U 1898, 8995,3C  ;19675          JSR [BACK.T31.LOOP]             ;WRITE ALL 1'S IN 4 LONGWORDS STARTING
                 ;19676                                          ; AT PHYSICAL ADDRESS 200
                 ;19677  LOOP.T31.3:
U 1899, 1F9C,F5  ;19678          MEM.REQ[OCTA.WR.V.WCHK] ADRS[#0] DT[LONG]
                 ;19679                                          ;SET UP TO WRITE MEMORY ADDRESS 200-20FF(H)
U 189A, 3240,15  ;19680          WRITE.MEM LS[#1]                ;WRITE A 1 AT LONGWORD ADDRESS 200
U 189B, B242,15  ;19681          WRITE.MEM LS[#2]                ;WRITE A 2 AT LONGWORD ADDRESS 204
U 189C, B244,15  ;19682          WRITE.MEM LS[#4]                ;WRITE A 4 AT LONGWORD ADDRESS 208
U 189D, 3246,15  ;19683          WRITE.MEM LS[#8]                ;WRITE A 8 AT LONGWORD ADDRESS 20C
                 ;19684
U 189E, E5F8,15  ;19685          CLR LS[OS]                      ;CLEAR INDEX
U 189F, B652,15  ;19686          MOV LS[#200] TO WR[0]           ;200 IN WR
U 18A0, BE88,15  ;19687          MOV WR[0] TO LS[ADDRESS.DATA]   ;ADDRESS FOR PRINTOUT (START AT 200)
U 18A1, BE12,15  ;19688          MOV WR[0] TO LS[T9]             ;START READING AT ADDRESS 200
                 ;19689  READ.T31.3:
U 18A2, 36F2,95  ;19690          MOV LS[SHIFT.OS(3-0)] TO WR[1]  ;GET EXPECTED DATA
U 18A3, 1813,F5  ;19691          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                 ;19692                                          ;SET UP TO READ MEMORY
U 18A4, 3022,15  ;19693          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 18A5, 0869,3C  ;19694          JSR [CHECK.RESULT]              ;CHECK FOR CORRECT DATA
U 18A6, 8989,94  ;19695          JMP [LOOP.T31.3]                ;ERROR LOOP IF ENABLED
                 ;19696
U 18A7, 7FF8,15  ;19697          INC LS[OS]                      ;INCREMENT INDEX
U 18A8, EC13,15  ;19698          ADD WR[2] TO LS[T9]              ;ADD 4 TO MEMORY ADDRESS
```

```
U 18A9, EC89,15 ;19699          ADD WR[2] TO LS[ADDRESS.DATA]   ;ADD 4 TO ADDRESS TO PRINT
                ;19700          CMP WR[3] WITH LS[T9],          ;SEE IF DONE (WR 3 CONTAINS 210(H))
U 18AA, 4F13,B5 ;19701           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 18AB, 098A,21 ;19702          JMP.IF[NEQ] TO [READ.T31.3]     ;REPEAT UNTIL DONE
                ;19703
U 18AC, FF82,15 ;19704          INC LS[ERROR.NUMBER]            ;ERROR 4
U 18AD, B676,15 ;19705          MOV LS[MME] TO WR[0]            ;SET MME BIT IN WR
U 18AE, C77A,15 ;19706          BIS LS[TB.PAR.DIAG] TO WR[0]    ;SET TB PARITY DIAG BIT
U 18AF, 8A17,FC ;19707          JSR [WRITE.CSR1]                ;WRITE IN CSR
U 18B0, B652,15 ;19708          MOV LS[#200] TO WR[0]           ;ADDRESS 200 IN WR
U 18B1, BE12,15 ;19709          MOV WR[0] TO LS[T9]             ;ADDRESS IN LS
U 18B2, 8995,3C ;19710          JSR [BACK.T31.LOOP]             ;WRITE ALL 1'S IN 4 LONGWORDS STARTING
                ;19711                                          ; AT PHYSICAL ADDRESS 200
                ;19712  LOOP.T31.4:
U 18B3, 1F9C,F5 ;19713          MEM.REQ[OCTA.WR.V.WCHK] ADRS[#0] DT[LONG]
                ;19714                                          ;SET UP TO WRITE MEMORY ADDRESS 200-20FF(H)
U 18B4, 3240,15 ;19715          WRITE.MEM LS[#1]                ;WRITE A 1 AT LONGWORD ADDRESS 200
U 18B5, B242,15 ;19716          WRITE.MEM LS[#2]                ;WRITE A 2 AT LONGWORD ADDRESS 204
U 18B6, B244,15 ;19717          WRITE.MEM LS[#4]                ;WRITE A 4 AT LONGWORD ADDRESS 208
U 18B7, 3246,15 ;19718          WRITE.MEM LS[#8]                ;WRITE A 8 AT LONGWORD ADDRESS 20C
                ;19719
U 18B8, B652,15 ;19720          MOV LS[#200] TO WR[0]           ;200 IN WR
U 18B9, BE88,15 ;19721          MOV WR[0] TO LS[ADDRESS.DATA]   ;ADDRESS FOR PRINTOUT (START AT 200)
U 18BA, BE12,15 ;19722          MOV WR[0] TO LS[T9]             ;START READING AT ADDRESS 200
                ;19723  READ.T31.4:
U 18BB, 369E,95 ;19724          MOV LS[ONES] TO WR[1]           ;GET EXPECTED DATA
U 18BC, 1813,F5 ;19725          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;19726                                          ;SET UP TO READ MEMORY
U 18BD, 3022,15 ;19727          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 18BE, 0869,3C ;19728          JSR [CHECK.RESULT]              ;CHECK FOR CORRECT DATA
U 18BF, 098B,34 ;19729          JMP [LOOP.T31.4]                ;ERROR LOOP IF ENABLED
                ;19730
U 18C0, EC13,15 ;19731          ADD WR[2] TO LS[T9]             ;ADD 4 TO MEMORY ADDRESS
U 18C1, EC89,15 ;19732          ADD WR[2] TO LS[ADDRESS.DATA]   ;ADD 4 TO ADDRESS TO PRINT
                ;19733          CMP WR[3] WITH LS[T9],          ;SEE IF DONE (WR 3 CONTAINS 210(H))
U 18C2, 4F13,B5 ;19734           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 18C3, 898B,B1 ;19735          JMP.IF[NEQ] TO [READ.T31.4]     ;REPEAT UNTIL DONE
                ;19736
U 18C4, 0A17,DC ;19737          JSR [WRITE.CSR1.MME]            ;WRITE IN CSR1
U 18C5, B610,15 ;19738          MOV LS[T8] TO WR[0]             ;GET TB DATA
U 18C6, C57E,15 ;19739          BIC LS[BIT31] TO WR[0]          ;CLEAR VALID BIT
U 18C7, 3E10,15 ;19740          MOV WR[0] TO LS[T8]             ;PUT BACK IN LS FOR WRITE TO TB
U 18C8, 1852,F5 ;19741          MEM.REQ[WRITE.TB] ADRS[BIT9] DT[LONG]
                ;19742                                          ;SET UP TO WRITE TB ADDRESS 1
U 18C9, 3210,15 ;19743          WRITE.MEM LS[T8]                ;SET PROT C, MODIFY, AND BYTE OFFSET
                ;19744                                          ; BITS AND MAP VIRTUAL ADDRESS 0 TO
                ;19745                                          ; PHYSICAL ADDRESS 200 IN TB 1
U 18CA, 477E,15 ;19746          BIS LS[BIT31] TO WR[0]          ;SET VALID BIT
U 18CB, 4540,15 ;19747          BIC LS[BIT0] TO WR[0]           ;CLEAR ADDRESS BIT 0 (MAP VIRTUAL 0 TO
                ;19748                                          ; PHYSICAL 0)
U 18CC, 3E10,15 ;19749          MOV WR[0] TO LS[T8]             ;PUT DATA IN LS FOR WRITE TO TB
U 18CD, 989C,F5 ;19750          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;19751                                          ;SET UP TO WRITE TB ADDRESS 0
U 18CE, 3210,15 ;19752          WRITE.MEM LS[T8]                ;SET VALID,PROT C,MODIFY,AND BYTE OFFSET
                ;19753                                          ; BITS AND MAP VIRTUAL ADDRESS 0 TO
```

```
                ;19754                                  ; PHYSICAL ADDRESS 0 IN TB 0
U 18CF, B652,15 ;19755          MOV LS[#200] TO WR[0]   ;ADDRESS 200 IN WR
U 18D0, 4144,15 ;19756          SUB LS[#4] FROM WR[0]   ;ADDRESS 1FC(H) IN WR
U 18D1, BE12,15 ;19757          MOV WR[0] TO LS[T9]     ;ADDRESS IN LS
U 18D2, BE14,15 ;19758          MOV WR[0] TO LS[T10]    ;ALSO SAVE IN LS 10
U 18D3, 4145,95 ;19759          SUB LS[#4] FROM WR[3]   ;SET UP ENDING ADDRESS TO 20C
U 18D4, 8995,3C ;19760          JSR [BACK.T31.LOOP]     ;WRITE ALL 1'S IN 4 LONGWORDS STARTING
                ;19761                                  ; AT PHYSICAL ADDRESS 1FC(H)
                ;19762  LOOP.T31.5.6:
U 18D5, 3644,15 ;19763          MOV LS[#4] TO WR[0]     ;PUT 4 IN WR
U 18D6, 2040,15 ;19764          INC WR[0]               ;5 IN WR
U 18D7, BE82,15 ;19765          MOV WR[0] TO LS[ERROR.NUMBER]   ;ERROR 5
U 18D8, 1F14,F5 ;19766          MEM.REQ[OCTA.WR.V.WCHK] ADRS[T10] DT[LONG]
                ;19767                                  ;SET UP TO WRITE MEMORY ADDRESS 1FC-20B
U 18D9, 3240,15 ;19768          WRITE.MEM LS[#1]        ;WRITE A 1 AT LONGWORD ADDRESS 1FC
U 18DA, B242,15 ;19769          WRITE.MEM LS[#2]        ;WRITE A 2 AT LONGWORD ADDRESS 200
U 18DB, B244,15 ;19770          WRITE.MEM LS[#4]        ;WRITE A 4 AT LONGWORD ADDRESS 204
U 18DC, 3246,15 ;19771          WRITE.MEM LS[#8]        ;WRITE A 8 AT LONGWORD ADDRESS 208
                ;19772
U 18DD, 3614,15 ;19773          MOV LS[T10] TO WR[0]    ;1FC(H) IN WR
U 18DE, BE88,15 ;19774          MOV WR[0] TO LS[ADDRESS.DATA]   ;ADDRESS FOR PRINTOUT (START AT 1FC(H))
U 18DF, BE12,15 ;19775          MOV WR[0] TO LS[T9]     ;START READING AT ADDRESS 1FC(H)
U 18E0, 3640,95 ;19776          MOV LS[#1] TO WR[1]     ;GET EXPECTED DATA
U 18E1, 1813,F5 ;19777          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;19778                                  ;SET UP TO READ MEMORY
U 18E2, 3022,15 ;19779          MOV MEM.DATA TO WR[0]   ;GET RESULT
U 18E3, 0869,3C ;19780          JSR [CHECK.RESULT]      ;CHECK FOR CORRECT DATA
U 18E4, 098D,54 ;19781          JMP [LOOP.T31.5.6]      ;ERROR LOOP IF ENABLED
                ;19782
U 18E5, FF82,15 ;19783          INC LS[ERROR.NUMBER]    ;ERROR 6
U 18E6, EC13,15 ;19784          ADD WR[2] TO LS[T9]     ;ADD 4 TO MEMORY ADDRESS
U 18E7, EC89,15 ;19785          ADD WR[2] TO LS[ADDRESS.DATA]   ;ADD 4 TO ADDRESS TO PRINT
                ;19786  READ.T31.6:
U 18E8, 369E,95 ;19787          MOV LS[ONES] TO WR[1]   ;GET EXPECTED DATA (UNMODIFIED)
U 18E9, 1813,F5 ;19788          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;19789                                  ;SET UP TO READ MEMORY
U 18EA, 3022,15 ;19790          MOV MEM.DATA TO WR[0]   ;GET RESULT
U 18EB, 0869,3C ;19791          JSR [CHECK.RESULT]      ;CHECK FOR UNMODIFIED DATA (ERROR SUM
                ;19792                                  ; SHOULD HAVE ABORTED WRITE)
U 18EC, 098D,54 ;19793          JMP [LOOP.T31.5.6]      ;ERROR LOOP IF ENABLED
                ;19794
U 18ED, EC13,15 ;19795          ADD WR[2] TO LS[T9]     ;ADD 4 TO MEMORY ADDRESS
U 18EE, EC89,15 ;19796          ADD WR[2] TO LS[ADDRESS.DATA]   ;ADD 4 TO ADDRESS TO PRINT
                ;19797          CMP WR[3] WITH LS[T9],  ;SEE IF DONE (WR 3 CONTAINS 20C(H))
U 18EF, 4F13,B5 ;19798           DT(LONG)&SET.ALU.CC    ; SET CONDITION CODES
U 18F0, 898E,81 ;19799          JMP.IF[NEQ] TO [READ.T31.6]     ;REPEAT UNTIL DONE
                ;19800
U 18F1, 3614,15 ;19801          MOV LS[T10] TO WR[0]    ;ADDRESS 1FC(H) IN WR
U 18F2, 4144,15 ;19802          SUB LS[#4] FROM WR[0]   ;ADDRESS 1F8(H) IN WR
U 18F3, BE12,15 ;19803          MOV WR[0] TO LS[T9]     ;ADDRESS IN LS
U 18F4, BE14,15 ;19804          MOV WR[0] TO LS[T10]    ;ALSO SAVE IN LS 10
U 18F5, 4145,95 ;19805          SUB LS[#4] FROM WR[3]   ;SET UP ENDING ADDRESS TO 208
U 18F6, 8995,3C ;19806          JSR [BACK.T31.LOOP]     ;WRITE ALL 1'S IN 4 LONGWORDS STARTING
                ;19807                                  ; AT PHYSICAL ADDRESS 1F8(H)
                ;19808  LOOP.T31.7.8:
```

```
U 18F7, B646,15 ;19809          MOV LS[#8] TO WR[0]              ;PUT 8 IN WR
U 18F8, 2100,15 ;19810          DEC WR[0]                        ;7 IN WR
U 18F9, BE82,15 ;19811          MOV WR[0] TO LS[ERROR.NUMBER]    ;ERROR 7
U 18FA, 1F14,F5 ;19812          MEM.REQ[OCTA.WR.V.WCHK] ADRS[T10] DT[LONG]
                ;19813                                           ;SET UP TO WRITE MEMORY ADDRESS 1F8-207
U 18FB, 3240,15 ;19814          WRITE.MEM LS[#1]                 ;WRITE A 1 AT LONGWORD ADDRESS 1F8
U 18FC, B242,15 ;19815          WRITE.MEM LS[#2]                 ;WRITE A 2 AT LONGWORD ADDRESS 1FC
U 18FD, B244,15 ;19816          WRITE.MEM LS[#4]                 ;WRITE A 4 AT LONGWORD ADDRESS 200
U 18FE, 3246,15 ;19817          WRITE.MEM LS[#8]                 ;WRITE A 8 AT LONGWORD ADDRESS 204
                ;19818
U 18FF, 3614,15 ;19819          MOV LS[T10] TO WR[0]             ;1F8(H) IN WR
U 1900, BE88,15 ;19820          MOV WR[0] TO LS[ADDRESS.DATA]    ;ADDRESS FOR PRINTOUT (START AT 1F8(H))
U 1901, BE12,15 ;19821          MOV WR[0] TO LS[T9]              ;START READING AT ADDRESS 1F8(H)
U 1902, 3640,95 ;19822          MOV LS[#1] TO WR[1]              ;GET EXPECTED DATA
U 1903, 1813,F5 ;19823          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;19824                                           ;SET UP TO READ MEMORY
U 1904, 3022,15 ;19825          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 1905, 0869,3C ;19826          JSR [CHECK.RESULT]               ;CHECK FOR CORRECT DATA
U 1906, 098F,74 ;19827          JMP [LOOP.T31.7.8]               ;ERROR LOOP IF ENABLED
                ;19828
U 1907, EC13,15 ;19829          ADD WR[2] TO LS[T9]              ;ADD 4 TO MEMORY ADDRESS
U 1908, EC89,15 ;19830          ADD WR[2] TO LS[ADDRESS.DATA]    ;ADD 4 TO ADDRESS TO PRINT
U 1909, B642,95 ;19831          MOV LS[#2] TO WR[1]              ;GET EXPECTED DATA
U 190A, 1813,F5 ;19832          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;19833                                           ;SET UP TO READ MEMORY
U 190B, 3022,15 ;19834          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 190C, 0869,3C ;19835          JSR [CHECK.RESULT]               ;CHECK FOR CORRECT DATA
U 190D, 098F,74 ;19836          JMP [LOOP.T31.7.8]               ;ERROR LOOP IF ENABLED
                ;19837
U 190E, FF82,15 ;19838          INC LS[ERROR.NUMBER]             ;ERROR 8
U 190F, EC13,15 ;19839          ADD WR[2] TO LS[T9]              ;ADD 4 TO MEMORY ADDRESS
U 1910, EC89,15 ;19840          ADD WR[2] TO LS[ADDRESS.DATA]    ;ADD 4 TO ADDRESS TO PRINT
                ;19841  READ.T31.8:
U 1911, 369E,95 ;19842          MOV LS[ONES] TO WR[1]            ;GET EXPECTED DATA (UNMODIFIED)
U 1912, 1813,F5 ;19843          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;19844                                           ;SET UP TO READ MEMORY
U 1913, 3022,15 ;19845          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 1914, 0869,3C ;19846          JSR [CHECK.RESULT]               ;CHECK FOR UNMODIFIED DATA (ERROR SUM
                ;19847                                           ; SHOULD HAVE ABORTED WRITE)
U 1915, 098F,74 ;19848          JMP [LOOP.T31.7.8]               ;ERROR LOOP IF ENABLED
                ;19849
U 1916, EC13,15 ;19850          ADD WR[2] TO LS[T9]              ;ADD 4 TO MEMORY ADDRESS
U 1917, EC89,15 ;19851          ADD WR[2] TO LS[ADDRESS.DATA]    ;ADD 4 TO ADDRESS TO PRINT
                ;19852          CMP WR[3] WITH LS[T9],           ;SEE IF DONE (WR 3 CONTAINS 20C(H))
U 1918, 4F13,B5 ;19853           DT(LONG)&SET.ALU.CC             ; SET CONDITION CODES
U 1919, 0991,11 ;19854          JMP.IF[NEQ] TO [READ.T31.8]      ;REPEAT UNTIL DONE
                ;19855
U 191A, FF82,15 ;19856          INC LS[ERROR.NUMBER]             ;ERROR 9
U 191B, B610,15 ;19857          MOV LS[T8] TO WR[0]              ;GET TB DATA
U 191C, C740,15 ;19858          BIS LS[BIT0] TO WR[0]            ;SET ADDRESS BIT 0 (MAP VIRTUAL 0 TO
                ;19859                                           ; PHYSICAL 200)
U 191D, 3E10,15 ;19860          MOV WR[0] TO LS[T8]              ;PUT DATA IN LS FOR WRITE TO TB
U 191E, 1852,F5 ;19861          MEM.REQ[WRITE.TB] ADRS[BIT9] DT[LONG]
                ;19862                                           ;SET UP TO WRITE TB ADDRESS 1
U 191F, 3210,15 ;19863          WRITE.MEM LS[T8]                 ;SET VALID,PROT C,MODIFY,AND BYTE OFFSET
```

```
                ;19864                                  ; BITS AND MAP VIRTUAL ADDRESS 0 TO
                ;19865                                  ; PHYSICAL ADDRESS 200 IN TB 1
U 1920, B652,15 ;19866          MOV LS[#200] TO WR[0]   ;ADDRESS 200 IN WR
U 1921, 4144,15 ;19867          SUB LS[#4] FROM WR[0]   ;ADDRESS 1FC(H) IN WR
U 1922, BE12,15 ;19868          MOV WR[0] TO LS[T9]     ;ADDRESS IN LS
U 1923, BE14,15 ;19869          MOV WR[0] TO LS[T10]    ;ALSO SAVE IN LS 10
U 1924, 4145,95 ;19870          SUB LS[#4] FROM WR[3]   ;SET UP ENDING ADDRESS TO 20C
U 1925, 8995,3C ;19871          JSR [BACK.T31.LOOP]     ;WRITE ALL 1'S IN 4 LONGWORDS STARTING
                ;19872                                  ; AT PHYSICAL ADDRESS 1FC(H)
                ;19873  LOOP.T31.9:
U 1926, 1F14,F5 ;19874          MEM.REQ[OCTA.WR.V.WCHK] ADRS[T10] DT[LONG]
                ;19875                                  ;SET UP TO WRITE MEMORY ADDRESS 1FC-20B
U 1927, 3240,15 ;19876          WRITE.MEM LS[#1]        ;WRITE A 1 AT LONGWORD ADDRESS 1FC
U 1928, B242,15 ;19877          WRITE.MEM LS[#2]        ;WRITE A 2 AT LONGWORD ADDRESS 200
U 1929, B244,15 ;19878          WRITE.MEM LS[#4]        ;WRITE A 4 AT LONGWORD ADDRESS 204
U 192A, 3246,15 ;19879          WRITE.MEM LS[#8]        ;WRITE A 8 AT LONGWORD ADDRESS 208
                ;19880
U 192B, 3614,15 ;19881          MOV LS[T10] TO WR[0]    ;1FC(H) IN WR
U 192C, BE88,15 ;19882          MOV WR[0] TO LS[ADDRESS.DATA]   ;ADDRESS FOR PRINTOUT (START AT 1FC(H))
U 192D, BE12,15 ;19883          MOV WR[0] TO LS[T9]     ;START READING AT ADDRESS 1FC(H)
U 192E, E5F8,15 ;19884          CLR LS[OS]              ;CLEAR INDEX
                ;19885  READ.T31.9:
U 192F, 36F2,95 ;19886          MOV LS[SHIFT.OS(3-0)] TO WR[1]  ;GET EXPECTED DATA
U 1930, 1813,F5 ;19887          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;19888                                  ;SET UP TO READ MEMORY
U 1931, 3022,15 ;19889          MOV MEM.DATA TO WR[0]   ;GET RESULT
U 1932, 0869,3C ;19890          JSR [CHECK.RESULT]      ;CHECK FOR CORRECT DATA
U 1933, 8992,64 ;19891          JMP [LOOP.T31.9]        ;ERROR LOOP IF ENABLED
                ;19892
U 1934, 7FF8,15 ;19893          INC LS[OS]              ;INCREMENT INDEX
U 1935, EC13,15 ;19894          ADD WR[2] TO LS[T9]     ;ADD 4 TO MEMORY ADDRESS
U 1936, EC89,15 ;19895          ADD WR[2] TO LS[ADDRESS.DATA]   ;ADD 4 TO ADDRESS TO PRINT
                ;19896          CMP WR[3] WITH LS[T9],  ;SEE IF DONE (WR 3 CONTAINS 210(H))
U 1937, 4F13,B5 ;19897           DT(LONG)&SET.ALU.CC    ; SET CONDITION CODES
U 1938, 8992,F1 ;19898          JMP.IF[NEQ] TO [READ.T31.9]     ;REPEAT UNTIL DONE
                ;19899
U 1939, FF82,15 ;19900          INC LS[ERROR.NUMBER]    ;ERROR A
U 193A, 3614,15 ;19901          MOV LS[T10] TO WR[0]    ;ADDRESS 1FC(H) IN WR
U 193B, BE12,15 ;19902          MOV WR[0] TO LS[T9]     ;ADDRESS IN LS
U 193C, 8995,3C ;19903          JSR [BACK.T31.LOOP]     ;WRITE ALL 1'S IN 4 LONGWORDS STARTING
                ;19904                                  ; AT PHYSICAL ADDRESS 1FC(H)
                ;19905  LOOP.T31.A:
U 193D, 1F14,F5 ;19906          MEM.REQ[OCTA.WR.V.WCHK] ADRS[T10] DT[LONG]
                ;19907                                  ;SET UP TO WRITE MEMORY ADDRESS 1FC-20B
U 193E, 3240,15 ;19908          WRITE.MEM LS[#1]        ;WRITE A 1 AT LONGWORD ADDRESS 1FC
U 193F, 0A1B,4C ;19909          JSR [NOP.DELAY]         ;EXECUTE 5 CYCLE DELAY TO DELAY CPU DR
U 1940, B242,15 ;19910          WRITE.MEM LS[#2]        ;WRITE A 2 AT LONGWORD ADDRESS 200
U 1941, B244,15 ;19911          WRITE.MEM LS[#4]        ;WRITE A 4 AT LONGWORD ADDRESS 204
U 1942, 3246,15 ;19912          WRITE.MEM LS[#8]        ;WRITE A 8 AT LONGWORD ADDRESS 208
                ;19913
U 1943, 3614,15 ;19914          MOV LS[T10] TO WR[0]    ;1FC(H) IN WR
U 1944, BE88,15 ;19915          MOV WR[0] TO LS[ADDRESS.DATA]   ;ADDRESS FOR PRINTOUT (START AT 1FC(H))
U 1945, BE12,15 ;19916          MOV WR[0] TO LS[T9]     ;START READING AT ADDRESS 1FC(H)
U 1946, E5F8,15 ;19917          CLR LS[OS]              ;CLEAR INDEX
                ;19918  READ.T31.A:
```

```
U 1947, 36F2,95 ;19919          MOV LS[SHIFT.OS(3-0)] TO WR[1]  ;GET EXPECTED DATA
U 1948, 1813,F5 ;19920          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;19921                                          ;SET UP TO READ MEMORY
U 1949, 3022,15 ;19922          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 194A, 0869,3C ;19923          JSR [CHECK.RESULT]              ;CHECK FOR CORRECT DATA
U 194B, 8993,D4 ;19924          JMP [LOOP.T31.A]                ;ERROR LOOP IF ENABLED
                ;19925
U 194C, 7FF8,15 ;19926          INC LS[OS]                      ;INCREMENT INDEX
U 194D, EC13,15 ;19927          ADD WR[2] TO LS[T9]             ;ADD 4 TO MEMORY ADDRESS
U 194E, EC89,15 ;19928          ADD WR[2] TO LS[ADDRESS.DATA]   ;ADD 4 TO ADDRESS TO PRINT
                ;19929          CMP WR[3] WITH LS[T9],          ;SEE IF DONE (WR 3 CONTAINS 210(H))
U 194F, 4F13,B5 ;19930           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 1950, 0994,71 ;19931          JMP.IF[NEQ] TO [READ.T31.A]     ;REPEAT UNTIL DONE
                ;19932
U 1951, 0995,94 ;19933          JMP [END.T31]                   ;DONE
                ;19934
                ;19935  ;SUBROUTINE TO WRITE BACKGROUND OF 1'S IN 4 LONGWORDS OF MEMORY
                ;19936
                ;19937  BACK.T31:
U 1952, 6512,15 ;19938          CLR LS[T9]                      ;START AT ADDRESS 0
                ;19939  BACK.T31.LOOP:
U 1953, 9912,75 ;19940          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;19941                                          ;SET UP TO WRITE 4 LONGWORDS
U 1954, 329E,15 ;19942          WRITE.MEM LS[ONES]              ;WRITE DATA OF ALL 1'S IN MEMORY
U 1955, EC13,15 ;19943          ADD WR[2] TO LS[T9]             ;ADD 4 TO MEMORY ADDRESS
                ;19944          CMP WR[3] WITH LS[T9],          ;SEE IF DONE
U 1956, 4F13,B5 ;19945           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 1957, 0995,31 ;19946          JMP.IF[NEQ] TO [BACK.T31.LOOP]  ;REPEAT UNTIL DONE
U 1958, 5B00,14 ;19947          RETURN                          ;DONE
                ;19948
                ;19949
                ;19950  END.T31:
```

```
;19951  .page   "  TEST 32 READ QUAD and READ OCTA Flow  (M8391 MCT Module)"
;19952  ;++
;19953  ;
;19954  ; Test Description:
;19955  ;
;19956  ;       This test is designed to check for faults in the control store PROMs
;19957  ;       by verifying all paths of the micro-code.  This test checks the path
;19958  ;       taken when a QUAD.READ.V.RCHK or OCTA READ is issued.  This test will
;19959  ;       check aligned and unaligned reads as well as reading locations with
;19960  ;       single or double bit errors.
;19961  ;         The first part of the test simply writes all ones, all zeros, and all
;19962  ;       ones pattern in 3 locations of memory (200 through 20B).  The QUAD.READ
;19963  ;       .V.RCHK is executed and the result checked.  This test is repeated with
;19964  ;       a delay of 5 NOPs between the MEM.REQ and each MOV MEM.DATA to delay
;19965  ;       CPU DR.  This will test the branch path on CPU DR not.
;19966  ;         The second part of the test checks the path when an error sum occurs
;19967  ;       on the first cycle.  Error sum is forced by setting TB PAR DIAG in CSR
;19968  ;       1 prior to executing the test.  The first cycle should complete but the
;19969  ;       second cycle should be aborted.
;19970  ;         The third part of the test checks the QUAD.READ.V.RCHK on a unalign-
;19971  ;       ed boundary.  Addresses 202 through 209 are read.  This is repeated
;19972  ;       setting TB PAR DIAG in CSR 1 to force an error.  In this case, address
;19973  ;       200 is read before the flow is aborted.
;19974  ;         The OCTA READ flow is checked next using the aligned read first.  This
;19975  ;       is repeated with a delay between MOV MEM.DATA instructions to delay CPU
;19976  ;       DR.  Then the unaligned read is checked starting at address 202 (virtual
;19977  ;       address 2).
;19978  ;         The rest of the test checks that ERR and ERRSUM path at various points
;19979  ;       throughout the micro-flow path.  The last test checks OCTA.READ.P.
;19980  ;
;19981  ; Assumptions:
;19982  ;
;19983  ;       It is assumed that all previous tests have run successfully, and
;19984  ;       that all hardware (other than MCT control store PROMs) is operating
;19985  ;       correctly.
;19986  ;
;19987  ; Test Steps:
;19988  ;
;19989  ;       1)  Set up error mask, error number, and module number in LS (for
;19990  ;           error printouts) and clear previous error number in LS.
;19991  ;       2)  Set up TB to map virtual address 0 to physical address 200.  Set
;19992  ;           MME bit in CSR 1.  Write a background of all ones, all zeros, all
;19993  ;           ones, all zeros starting at address 200.
;19994  ;       3)  Execute QUAD.READ.V.RCHK and verify correct data reads.
;19995  ;       4)  Repeat steps 2 and 3 with delays between the MEM.REQ and the MOV
;19996  ;           MEM.DATA TO LS instructions to force the CPU DR not path in the
;19997  ;           micro flows.
;19998  ;       5)  Set both MME and TB PAR DIAG bits in CSR 1.  Execute QUAD.READ.V
;19999  ;           .RCHK and check that first read cycle completes and that second
;20000  ;           read cycle aborts due to ERROR SUM.
;20001  ;       6)  Set MME bit only in CSR 1.  Execute QUAD.READ.V.RCHK on virtual ad-
;20002  ;           dress 2 (unaligned read) and verify that both longword reads are
;20003  ;           correct.
;20004  ;       7)  Repeat step 6 with delayed CPU DR.
;20005  ;       8)  Set both MME and TB PAR DIAG bits in CSR 1 and repeat read in step
```

```
;20006  ;          6.  Verify that address 200 (the first address read in an unaligned
;20007  ;          read) is read correctly and that the following read cycles are
;20008  ;          aborted due to ERROR SUM.
;20009  ;       9) Set MME bit in CSR 1.  Execute OCTA.RD.V.RCHK starting at address
;20010  ;          200.  Check all four longwords for correct data.
;20011  ;       10) Repeat step 9 with a delay between the MOV MEM.DATA instructions to
;20012  ;          take the CPU DR not path in the microcode.
;20013  ;       11) Repeat step 9 starting at address 202 (unaligned read).  This checks
;20014  ;          the 2 MEM CYC and ICO not path in the micro-code.
;20015  ;       12) Repeat step 11 with delayed CPU DR.
;20016  ;       13) Write single bit errors in five longwords starting at address 200.
;20017  ;       14) Execute OCTA.READ.V.RCHK on unaligned boundary and check that each
;20018  ;          longword has the error corrected and is read correctly.
;20019  ;       15) Set up a double bit error at address 200.  Execute QUAD.RD.V.RCHK
;20020  ;          at address 202 and verify that an abort occurs.
;20021  ;       16) Write good data at 200, set up a double bit error at address 204.
;20022  ;          Execute QUAD.RD.V.RCHK and check for abort on second longword.
;20023  ;       17) Write good data at 204, set up a double bit error at address 208.
;20024  ;          Execute OCTA.RD.V.RCHK and check for abort on third longword.
;20025  ;       18) Set up TB to map virtual 0 to physical 0 in TB 0.  Set up TB 1 to
;20026  ;          map virtual 0 to physical 200 and clear the VALID bit.  This will
;20027  ;          cause an error sum when TB 1 is used.
;20028  ;       19) Write good data at address 1FC.  Execute QUAD.RD.V.RCHK at address
;20029  ;          1FE and verify that abort occurs on second longword.
;20030  ;       20) Write good data at address 1F8.  Execute OCTA.RD.V.RCHK at address
;20031  ;          1FA and verify that abort occurs on third longword.
;20032  ;       21) Write 4 longwords of good data at address 200.  Execute OCTA.READ.P
;20033  ;          and verify correct data comes back.  This check entry point for
;20034  ;          physical read.
;20035  ;
;20036  ; Errors:
;20037  ;
;20038  ;       NOTE:  Data printed under OTHER in error report is address of failing
;20039  ;       location.
;20040  ;
;20041  ;       Error 1 - QUAD.READ.V.RCHK failed to read 2 longwords correctly (CPU DR
;20042  ;                 was not delayed).
;20043  ;       Error 2 - QUAD.READ.V.RCHK failed to read 2 longwords correctly (CPU DR
;20044  ;                 delayed).
;20045  ;       Error 3 - QUAD.READ.V.RCHK failed to read first cycle correctly (200)
;20046  ;                 or failed to abort second read cycle (204).  Address under
;20047  ;                 other indicates which cycle failed.
;20048  ;       Error 4 - QUAD.READ.V.RCHK failed on unaligned read.
;20049  ;       Error 5 - QUAD.READ.V.RCHK failed on unaligned read with delayed CPU DR.
;20050  ;       Error 6 - QUAD.READ.V.RCHK failed to abort second read cycle (204) on
;20051  ;                 unaligned read with TB PAR ERR.
;20052  ;       Error 7 - OCTA.RD.V.RCHK failed.
;20053  ;       Error 8 - OCTA.RD.V.RCHK failed with delayed CPU DR.
;20054  ;       Error 9 - OCTA.RD.V.RCHK failed on unaligned read.
;20055  ;       Error A - OCTA.RD.V.RCHK failed on unaligned read with delayed CPU DR.
;20056  ;       Error B - OCTA.RD.V.RCHK failed on unaligned read with single bit
;20057  ;                 errors in each longword.
;20058  ;       Error C - QUAD.RD.V.RCHK failed to abort on unaligned read with uncor-
;20059  ;                 rectable error in first longword.
;20060  ;       Error D - QUAD.RD.V.RCHK failed to read first longword correctly (202)
```

```
;20061  ;                   or failed to abort second read (206) on unaligned read with
;20062  ;                   uncorrectable error in second longword.
;20063  ;       Error E - OCTA.RD.V.RCHK failed to read first or second longword cor-
;20064  ;                   rectly (202 or 206) or failed to abort third read (20A) on
;20065  ;                   unaligned read with uncorrectable error in third longword.
;20066  ;       Error F - QUAD.RD.V.RCHK failed to read first longword correctly (1FE)
;20067  ;                   or failed to abort second read (202) on unaligned read with
;20068  ;                   errorsum on second longword (TB not valid).
;20069  ;       Error 10- OCTA.RD.V.RCHK failed to read first or second longword cor-
;20070  ;                   rectly (1FA or 1FE) or failed to abort third read (202) on
;20071  ;                   unaligned read with errorsum on third longword (TB not valid).
;20072  ;       Error 11- OCTA.READ.P failed to read four longwords correctly.
;20073  ;
;20074  ; Debug:
;20075  ;
;20076  ;       Note:  The following errors most likely indicate a problem with the
;20077  ;       micro-code flow.  The hardware used by the micro-code has been pre-
;20078  ;       viously tested.  To trace a problem, check the sequence outlined
;20079  ;       below.  If the sequence is OK, suspect a bad bit in one of the control
;20080  ;       store PROMs and check each state for the correct enables.
;20081  ;
;20082  ;       Error 1 - This is the first time that an QUAD READ is tested, so any
;20083  ;       instruction in the micro flow could be causing a problem.  The path
;20084  ;       that should be observed is as follows:  QUAD READ V RCHK to OCTA ARY
;20085  ;       CYC C4 to OCTA C5 to OCTA C6 to OCTA C7 to OCTA C8 to OCTA C9 to
;20086  ;       OCTA C10A to OCTA C11 A to CPU RD V CONT C7.  From there the flow has
;20087  ;       been tested previously.
;20088  ;
;20089  ;       Error 2 - This error uses a different branch at OCTA C8.  The flow is
;20090  ;       the same as error 1 above until OCTA C8.  From there the L CPU DR not
;20091  ;       and P2 not path should be taken to the cycle that loops waiting for
;20092  ;       CPU DR.  Then to OCTA WAIT CPU DR C1A to OCTA WAIT REF A to the cycle
;20093  ;       that branches on ICO and 1 MEM CYC to OCTA C10A to OCTA C11A to CPU
;20094  ;       RD V CONT C7.
;20095  ;
;20096  ;       Error 3 - This error indicates a problem with the ERRSUM branch at
;20097  ;       OCTA C6.  The path taken at OCTA C6 should have gone the ERRSUM route
;20098  ;       to the cycle which goes to WAIT CPUDR C1.  The first read should have
;20099  ;       executed OK (at address 200).  The second read should have been aborted
;20100  ;       (at address 204).
;20101  ;
;20102  ;       Error 4 - This error indicates a problem with the path taken at OCTA
;20103  ;       ARY CYC C4.  The 2 MEM CYC path should be taken.  From here the flow
;20104  ;       is to OCTA TWO ARY CYC C5 to OCTA TWO ARY CYC C6 to OCTA TWO ARY CYC
;20105  ;       C7 to OCTA TWO ARY CYC C8 to the cycle that goes back to OCTA ARY CYC
;20106  ;       C4.  Again the 2 MEM CYC branch is taken to OCTA TWO ARY CYC C5.  The
;20107  ;       2ND CYC branch is taken to the cycle which checks for ERRSUM and then
;20108  ;       to OCTA C7.  From there the path is the same as in error 1.
;20109  ;
;20110  ;       Error 5 -  Same as error 4 except the CPU DR not and P2 not path is
;20111  ;       taken at OCTA C8.
;20112  ;
;20113  ;       Error 6 - This error should have taken the ERRSUM path at OCTA TWO ARY
;20114  ;       CYC C6.  The first cycle should have read the data at address 200 and
;20115  ;       put it on the bus before aborting (OTHER=200).  The second cycle should
```

```
;20116  ;       have been aborted so that LS 10 should have received the same data.
;20117  ;       The path at OCTA TWO ARY CYC C6 should have been to the cycle that goes
;20118  ;       to WAIT.CPUDR C1.
;20119  ;
;20120  ;       Error 7 - This error is the first to use the OCTA read flow which con-
;20121  ;       tinues after the QUAD flow has ended.  The flow starts at OCTA READ V
;20122  ;       .RCHK and goes to OCTA ARY CYC C4.  From there it proceeds as in error
;20123  ;       1 above until OCTA C9.  At this point the ICO not and 1 MEM CYC path is
;20124  ;       taken.  The flow goes to OCTA C11 after one cycle and then to OCTA C12
;20125  ;       to OCTA C13 to OCTA C14 to OCTA C15 to another cycle whichi goes to
;20126  ;       OCTA C17 back to OCTA C6.  From there the flow proceeds as error 1 from
;20127  ;       OCTA C6.
;20128  ;
;20129  ;       Error 8 - This error is similiar to error 6 above except the CPU DR not
;20130  ;       and P2 not path is taken at two points: OCTA C8 and OCTA C14.  These
;20131  ;       paths and the cycle which branches on ICO and 1 MEM CYCLE are the most
;20132  ;       suspect.
;20133  ;
;20134  ;       Error 9 - This error takes the same path initially (after OCTA READ V
;20135  ;       RCHK to OCTA ARY CYC C4) as error 4 above except the ICO not and 2 MEM
;20136  ;       CYC branch is taken at OCTA C9 and the 2 MEM CYC path is taken at OCTA
;20137  ;       C15.
;20138  ;
;20139  ;       Error A -  Same as error 9 except the CPU DR not and P2 not path is
;20140  ;       taken at OCTA C8.
;20141  ;
;20142  ;       Error B - This error is essentially the same as error 9 except that the
;20143  ;       ERR path is taken at OCTA TWO ARY CYC C6, OCTA C7, and OCTA C13.  Once
;20144  ;       in the error path, the SERR branch should be taken in all three cases
;20145  ;       and the error should be corrected.
;20146  ;
;20147  ;       Error C - This error checks the UNC ERR branch at OCTA 2 ARY CYC WSDE
;20148  ;       C9.  The flow up to that point is the same as error B.  At OCTA 2 ARY
;20149  ;       CYC WSDE C9 the UNC ERR branch should be taken to OCTA 2 ARY CYC WUDE
;20150  ;       C9.
;20151  ;
;20152  ;       Error D - This error should follow the same path as error 4 up to OCTA
;20153  ;       C7.  At that point, the ERR path should be taken to OCTA ERR C8.  From
;20154  ;       there the UNC ERR branch to OCTA ERR C9 should take place.
;20155  ;
;20156  ;       Error E - This error should follow the same path as error 9 up to OCTA
;20157  ;       C13.  At that point the ERR path should be taken to OCTA ERR C14.  From
;20158  ;       there the UNC ERR path should be taken to the cycle which goes to WAIT
;20159  ;       CPUDR C1.
;20160  ;
;20161  ;       Error F - This error checks the ERRSUM branch at the cycle after OCTA
;20162  ;       TWO ARY CYC C5 (2ND CYC path).  The flow is the same as error 4 up to
;20163  ;       that point.  Then the ERRSUM path should be taken to end up at WAIT
;20164  ;       CPUDR C1.
;20165  ;
;20166  ;       Error 10 - This error checks the ERRSUM branch at OCTA C12.  The flow
;20167  ;       is the same as error 9 up to that point.  Then the ERRSUM path that ends
;20168  ;       up at WAIT CPUDR C1 should be taken.
;20169  ;
;20170  ;       Error 11 - Same as error 7 except entry is at CPU READ PH OCTA.
```

```
                    :20171  ;--
                    :20172
                    :20173  T.32:
U 1959, B65E,15     :20174          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    :20175                                          ; STATUS WORD
U 195A, 3E80,15     :20176          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    :20177                                          ; BIT 15 INDICATES START OF TEST TO
                    :20178                                          ; CONSOLE
U 195B, 10E0,15     :20179          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    :20180  WAIT.T32.0:
U 195C, 0995,C4     :20181          JMP [WAIT.T32.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    :20182                                          ; RESPOND
                    :20183
U 195D, 0A1A,AC     :20184          JSR [SETUP.1]                   ;SET UP MASKS, MODULE CODE ETC.
                    :20185
U 195E, 367E,15     :20186          MOV LS[BIT31] TO WR[0]          ;SET BIT 31 IN WR
U 195F, C77A,15     :20187          BIS LS[BIT29] TO WR[0]          ;AND BIT 29
U 1960, 4772,15     :20188          BIS LS[BIT25] TO WR[0]          ;AND BIT 25
U 1961, C740,15     :20189          BIS LS[BIT0] TO WR[0]           ;AND BIT 0
U 1962, 3E10,15     :20190          MOV WR[0] TO LS[T8]             ;PUT IN TEMP LS
U 1963, 989C,F5     :20191          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                    :20192                                          ;SET UP TO WRITE TB ADDRESS 0
U 1964, 3210,15     :20193          WRITE.MEM LS[T8]                ;SET VALID, PROT C, AND BYTE OFFSET
                    :20194                                          ; BITS AND MAP VIRTUAL ADDRESS 0 TO
                    :20195                                          ; PHYSICAL ADDRESS 200 IN TB
U 1965, 0A17,DC     :20196          JSR [WRITE.CSR1.MME]            ;WRITE IN CSR 1
U 1966, B652,15     :20197          MOV LS[#200] TO WR[0]           ;GET 200 IN WR
U 1967, C044,15     :20198          ADD LS[#4] TO WR[0]             ;204 IN WR
U 1968, BE12,15     :20199          MOV WR[0] TO LS[T9]             ;STORE IN LS
U 1969, 1C52,75     :20200          MEM.REQ[OCTA.WRITE.P] ADRS[#200] DT[LONG]
                    :20201                                          ;SET UP TO WRITE 4 LONGWORDS IN MEMORY
U 196A, 329E,15     :20202          WRITE.MEM LS[ONES]              ;WRITE ALL ONES PATTERN AT 200
U 196B, B29C,15     :20203          WRITE.MEM LS[ZERO]              ;WRITE ALL ZEROS PATTERN AT 204
U 196C, 329E,15     :20204          WRITE.MEM LS[ONES]              ;WRITE ALL ONES PATTERN AT 208
U 196D, B29C,15     :20205          WRITE.MEM LS[ZERO]              ;WRITE ALL ZEROS PATTERN AT 20C
U 196E, B652,15     :20206          MOV LS[#200] TO WR[0]           ;200 IN WR
U 196F, C048,15     :20207          ADD LS[#10] TO WR[0]            ;210 IN WR
U 1970, BE14,15     :20208          MOV WR[0] TO LS[T10]            ;MEMORY ADDRESS TO WRITE IN LS
U 1971, 9914,75     :20209          MEM.REQ[WRITE.P] ADRS[T10] DT[LONG]
                    :20210                                          ;SET UP TO WRITE ADDRESS 210
U 1972, 329E,15     :20211          WRITE.MEM LS[ONES]              ;WRITE ALL ONES PATTERN AT 210
                    :20212
U 1973, B670,15     :20213          MOV LS[OTHER.DATA] TO WR[0]     ;SET BIT TO PRINT UNDER OTHER IN ERROR
                    :20214                                          ; REPORT
U 1974, 3E80,15     :20215          MOV WR[0] TO LS[CONTROL.STATUS] ;PUT IN CONTROL AND STATUS WORD
U 1975, 3653,15     :20216          MOV LS[#200] TO WR[2]           ;ADDRESS FOR OTHER DATA
U 1976, 3613,95     :20217          MOV LS[T9] TO WR[3]             ;NEXT ADDRESS FOR OTHER DATA
U 1977, E51C,15     :20218          CLR LS[T14]                     ;0 IN LS
U 1978, FD1C,15     :20219          COM LS[T14]                     ;EXPECTED DATA FOR 1ST LONGWORD
U 1979, 651E,15     :20220          CLR LS[T15]                     ;EXPECTED DATA FOR 2ND LONGWORD
                    :20221  LOOP.T32.1:
U 197A, 1F9D,75     :20222          MEM.REQ[QUAD.READ.V.RCHK] ADRS[#0] DT[LONG]
                    :20223                                          ;SET UP TO READ MEMORY ADDRESS 200-207
U 197B, 3022,15     :20224          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 197C, B814,15     :20225          MOV MEM.DATA TO LS[T10]         ;GET RESULT OF SECOND READ CYCLE
```

```
                    ;20226
U 197D, 09A9,4C     ;20227          JSR [CHECK.T32.QUAD]            ;CHECK RESULTS
U 197E, 8997,A4     ;20228          JMP [LOOP.T32.1]                ;ERROR LOOP IF ENABLED
                    ;20229
U 197F, FF82,15     ;20230          INC LS[ERROR.NUMBER]            ;ERROR 2
                    ;20231  LOOP.T32.2:
U 1980, 1F9D,75     ;20232          MEM.REQ[QUAD.READ.V.RCHK] ADRS[#0] DT[LONG]
                    ;20233                                          ;SET UP TO READ MEMORY ADDRESS 200-207
U 1981, 3022,15     ;20234          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 1982, 0A1B,4C     ;20235          JSR [NOP.DELAY]                 ;EXECUTE 5 CYCLE DELAY TO DELAY CPUDR
U 1983, B814,15     ;20236          MOV MEM.DATA TO LS[T10]          ;GET RESULT OF SECOND READ CYCLE
                    ;20237
U 1984, 09A9,4C     ;20238          JSR [CHECK.T32.QUAD]            ;CHECK RESULTS
U 1985, 8998,04     ;20239          JMP [LOOP.T32.2]                ;ERROR LOOP IF ENABLED
                    ;20240
U 1986, FF82,15     ;20241          INC LS[ERROR.NUMBER]            ;ERROR 3
U 1987, B676,15     ;20242          MOV LS[MME] TO WR[0]            ;SET MME BIT IN WR
U 1988, C77A,15     ;20243          BIS LS[TB.PAR.DIAG] TO WR[0]    ;SET TB PARITY DIAG BIT IN WR
U 1989, 8A17,FC     ;20244          JSR [WRITE.CSR1]                ;SET MME AND TB PARITY DIAG IN CSR 1
                    ;20245                                          ; (FORCE ERRSUM ON READ)
U 198A, 3645,95     ;20246          MOV LS[#4] TO WR[3]              ;INC ADDRESS BY 4 FOR SECOND LONGWORD
                    ;20247  LOOP.T32.3:
U 198B, 1F9D,75     ;20248          MEM.REQ[QUAD.READ.V.RCHK] ADRS[#0] DT[LONG]
                    ;20249                                          ;SET UP TO READ MEMORY ADDRESS 200-207
U 198C, 3022,15     ;20250          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 198D, B814,15     ;20251          MOV MEM.DATA TO LS[T10]          ;SECOND CYCLE SHOULD BE ABORTED
                    ;20252
U 198E, 3E89,15     ;20253          MOV WR[2] TO LS[ADDRESS.DATA]    ;ADDRESS FOR PRINTOUT
U 198F, 369E,95     ;20254          MOV LS[ONES] TO WR[1]            ;GET EXPECTED DATA
U 1990, 0869,3C     ;20255          JSR [CHECK.RESULT]              ;CHECK FOR CORRECT DATA
U 1991, 0998,B4     ;20256          JMP [LOOP.T32.3]                ;ERROR LOOP IF ENABLED
                    ;20257
U 1992, 2F82,95     ;20258          CLR WR[1]                       ;EXPECTED DATA IF ABORT FAILS (ADDR 204)
U 1993, 3614,15     ;20259          MOV LS[T10] TO WR[0]            ;GET SECOND LONGWORD
U 1994, 89AB,2C     ;20260          JSR [CHECK.T32.ABORT]           ;CHECK FOR ABORT
U 1995, 0998,B4     ;20261          JMP [LOOP.T32.3]                ;ERROR LOOP IF ENABLED
                    ;20262
U 1996, FF82,15     ;20263          INC LS[ERROR.NUMBER]            ;ERROR 4
U 1997, 0A17,DC     ;20264          JSR [WRITE.CSR1.MME]            ;SET MME IN CSR 1 (NO ERROR)
U 1998, 4043,15     ;20265          ADD LS[#2] TO WR[2]              ;202 FOR ADDRESS PRINTOUT
U 1999, 2005,95     ;20266          MOV WR[2] TO WR[3]               ;202 IN WR 3
U 199A, C045,95     ;20267          ADD LS[#4] TO WR[3]              ;206 FOR SECOND ADDRESS PRINOUT
U 199B, 3628,15     ;20268          MOV LS[#FFFF] TO WR[0]           ;GET FFFF IN WR
U 199C, 3E1C,15     ;20269          MOV WR[0] TO LS[T14]             ;EXPECTED DATA FOR 1ST LONGWORD (0000FFFF)
U 199D, A0C0,15     ;20270          COM WR[0]                       ;FFFF0000 IN WR
U 199E, BE1E,15     ;20271          MOV WR[0] TO LS[T15]             ;EXPECTED DATA FOR 2ND LONGWORD (FFFF0000)
                    ;20272  LOOP.T32.4:
U 199F, 1F43,75     ;20273          MEM.REQ[QUAD.READ.V.RCHK] ADRS[#2] DT[LONG]
                    ;20274                                          ;SET UP TO READ MEMORY ADDRESS 202-209
U 19A0, 3022,15     ;20275          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 19A1, B814,15     ;20276          MOV MEM.DATA TO LS[T10]          ;GET RESULT OF SECOND READ CYCLE
                    ;20277
U 19A2, 09A9,4C     ;20278          JSR [CHECK.T32.QUAD]            ;CHECK RESULTS
U 19A3, 0999,F4     ;20279          JMP [LOOP.T32.4]                ;ERROR LOOP IF ENABLED
                    ;20280
```

```
U 19A4, FF82,15 ;20281          INC LS[ERROR.NUMBER]              ;ERROR 5
                ;20282  LOOP.T32.5:
U 19A5, 1F43,75 ;20283          MEM.REQ[QUAD.READ.V.RCHK] ADRS[#2] DT[LONG]
                ;20284                                           ;SET UP TO READ MEMORY ADDRESS 202-209
U 19A6, 3022,15 ;20285          MOV MEM.DATA TO WR[0]             ;GET RESULT
U 19A7, 0A1B,4C ;20286          JSR [NOP.DELAY]                   ;EXECUTE 5 CYCLE DELAY TO DELAY CPU DR
U 19A8, B814,15 ;20287          MOV MEM.DATA TO LS[T10]           ;GET RESULT OF SECOND READ CYCLE
                ;20288
U 19A9, 09A9,4C ;20289          JSR [CHECK.T32.QUAD]              ;CHECK RESULTS
U 19AA, 099A,54 ;20290          JMP [LOOP.T32.5]                  ;ERROR LOOP IF ENABLED
                ;20291
U 19AB, FF82,15 ;20292          INC LS[ERROR.NUMBER]              ;ERROR 6
U 19AC, B676,15 ;20293          MOV LS[MME] TO WR[0]              ;SET MME BIT IN WR
U 19AD, C77A,15 ;20294          BIS LS[TB.PAR.DIAG] TO WR[0]      ;SET TB PARITY DIAG BIT IN WR
U 19AE, 8A17,FC ;20295          JSR [WRITE.CSR1]                  ;SET MME AND TB PARITY DIAG IN CSR 1
                ;20296                                           ; (FORCE ERRSUM ON READ)
U 19AF, 3653,15 ;20297          MOV LS[#200] TO WR[2]             ;ADDRESS FOR ERROR PRINTOUT
U 19B0, 4143,95 ;20298          SUB LS[#2] FROM WR[3]             ;FAKE ADDRESS FOR 2ND CYCLE
                ;20299  LOOP.T32.6:
U 19B1, 1F43,75 ;20300          MEM.REQ[QUAD.READ.V.RCHK] ADRS[#2] DT[LONG]
                ;20301                                           ;SET UP TO READ MEMORY ADDRESS 202-209
U 19B2, 3022,15 ;20302          MOV MEM.DATA TO WR[0]             ;GET RESULT (FIRST CYCLE SHOULD ONLY
                ;20303                                           ; READ ADDRESS 200)
U 19B3, B814,15 ;20304          MOV MEM.DATA TO LS[T10]           ;SECOND CYCLE SHOULD BE ABORTED
U 19B4, 369E,95 ;20305          MOV LS[ONES] TO WR[1]             ;GET EXPECTED DATA
U 19B5, 3E89,15 ;20306          MOV WR[2] TO LS[ADDRESS.DATA]     ;ADDRESS FOR PRINTOUT (200)
U 19B6, 0869,3C ;20307          JSR [CHECK.RESULT]                ;CHECK FOR CORRECT DATA
U 19B7, 099B,14 ;20308          JMP [LOOP.T32.6]                  ;ERROR LOOP IF ENABLED
                ;20309
U 19B8, FF82,15 ;20310          INC LS[ERROR.NUMBER]              ;ERROR 7
U 19B9, 0A17,DC ;20311          JSR [WRITE.CSR1.MME]              ;SET MME IN CSR 1
U 19BA, 3645,95 ;20312          MOV LS[#4] TO WR[3]               ;PUT 4 IN WR TO ADD TO ADDRESS
U 19BB, E51C,15 ;20313          CLR LS[T14]                       ;0 IN LS
U 19BC, FD1C,15 ;20314          COM LS[T14]                       ;ALL 1'S IN LS (EXPECTED DATA FOR FIRST
                ;20315                                           ; AND THIRD LONGWORD.  DATA FOR SECOND
                ;20316                                           ; AND FOURTH LONGWORD IS ALL 0'S)
                ;20317  LOOP.T32.7:
U 19BD, 9E9D,75 ;20318          MEM.REQ[OCTA.READ.V.RCHK] ADRS[#0] DT[LONG]
                ;20319                                           ;SET UP TO EXECUTE OCTA READ
U 19BE, 3022,15 ;20320          MOV MEM.DATA TO WR[0]             ;GET FIRST LONGWORD DATA
U 19BF, B814,15 ;20321          MOV MEM.DATA TO LS[T10]           ;GET SECOND LONGWORD DATA
U 19C0, 3816,15 ;20322          MOV MEM.DATA TO LS[T11]           ;GET THIRD LONGWORD DATA
U 19C1, B818,15 ;20323          MOV MEM.DATA TO LS[T12]           ;GET FOURTH LONGWORD DATA
                ;20324
U 19C2, 09A9,EC ;20325          JSR [CHECK.T32.OCTA]              ;CHECK RESULTS
U 19C3, 099B,D4 ;20326          JMP [LOOP.T32.7]                  ;ERROR LOOP IF ENABLED
                ;20327
U 19C4, FF82,15 ;20328          INC LS[ERROR.NUMBER]              ;ERROR 8
                ;20329  LOOP.T32.8:
U 19C5, 9E9D,75 ;20330          MEM.REQ[OCTA.READ.V.RCHK] ADRS[#0] DT[LONG]
                ;20331                                           ;SET UP TO EXECUTE OCTA READ
U 19C6, 3022,15 ;20332          MOV MEM.DATA TO WR[0]             ;GET FIRST LONGWORD DATA
U 19C7, 0A1B,4C ;20333          JSR [NOP.DELAY]                   ;EXECUTE 5 CYCLE DELAY TO DELAY CPU DR
U 19C8, B814,15 ;20334          MOV MEM.DATA TO LS[T10]           ;GET SECOND LONGWORD DATA
U 19C9, 0A1B,4C ;20335          JSR [NOP.DELAY]                   ;EXECUTE 5 CYCLE DELAY TO DELAY CPU DR
```

```
U 19CA, 3816,15 ;20336          MOV MEM.DATA TO LS[T11]          ;GET THIRD LONGWORD DATA
U 19CB, B818,15 ;20337          MOV MEM.DATA TO LS[T12]          ;GET FOURTH LONGWORD DATA
                ;20338
U 19CC, 09A9,EC ;20339          JSR [CHECK.T32.OCTA]             ;CHECK RESULTS
U 19CD, 099C,54 ;20340          JMP [LOOP.T32.8]                 ;ERROR LOOP IF ENABLED
                ;20341
U 19CE, FF82,15 ;20342          INC LS[ERROR.NUMBER]             ;ERROR 9
U 19CF, 4043,15 ;20343          ADD LS[#2] TO WR[2]              ;ADDRESS 202 IN WR FOR PRINTOUT
U 19D0, 3645,95 ;20344          MOV LS[#4] TO WR[3]              ;PUT 4 IN WR TO ADD TO ADDRESS
U 19D1, 3628,15 ;20345          MOV LS[#FFFF] TO WR[0]           ;FFFF IN WR
U 19D2, 3E1C,15 ;20346          MOV WR[0] TO LS[T14]             ;FFFF IN LS (EXPECTED DATA FOR FIRST
                ;20347                                           ; AND THIRD LONGWORD.  DATA FOR SECOND
                ;20348                                           ; AND FOURTH LONGWORD IS FFFF0000)
                ;20349  LOOP.T32.9:
U 19D3, 9E43,75 ;20350          MEM.REQ[OCTA.READ.V.RCHK] ADRS[#2] DT[LONG]
                ;20351                                           ;SET UP TO EXECUTE UNALIGNED OCTA READ
U 19D4, 3022,15 ;20352          MOV MEM.DATA TO WR[0]            ;GET FIRST LONGWORD DATA
U 19D5, B814,15 ;20353          MOV MEM.DATA TO LS[T10]          ;GET SECOND LONGWORD DATA
U 19D6, 3816,15 ;20354          MOV MEM.DATA TO LS[T11]          ;GET THIRD LONGWORD DATA
U 19D7, B818,15 ;20355          MOV MEM.DATA TO LS[T12]          ;GET FOURTH LONGWORD DATA
                ;20356
U 19D8, 09A9,EC ;20357          JSR [CHECK.T32.OCTA]             ;CHECK RESULTS
U 19D9, 899D,34 ;20358          JMP [LOOP.T32.9]                 ;ERROR LOOP IF ENABLED
                ;20359
U 19DA, FF82,15 ;20360          INC LS[ERROR.NUMBER]             ;ERROR A
                ;20361  LOOP.T32.A:
U 19DB, 9E43,75 ;20362          MEM.REQ[OCTA.READ.V.RCHK] ADRS[#2] DT[LONG]
                ;20363                                           ;SET UP TO EXECUTE UNALIGNED OCTA READ
U 19DC, 3022,15 ;20364          MOV MEM.DATA TO WR[0]            ;GET FIRST LONGWORD DATA
U 19DD, 0A18,4C ;20365          JSR [NOP.DELAY]                  ;EXECUTE 5 CYCLE DELAY TO DELAY CPU DR
U 19DE, B814,15 ;20366          MOV MEM.DATA TO LS[T10]          ;GET SECOND LONGWORD DATA
U 19DF, 3816,15 ;20367          MOV MEM.DATA TO LS[T11]          ;GET THIRD LONGWORD DATA
U 19E0, B818,15 ;20368          MOV MEM.DATA TO LS[T12]          ;GET FOURTH LONGWORD DATA
                ;20369
U 19E1, 09A9,EC ;20370          JSR [CHECK.T32.OCTA]             ;CHECK RESULTS
U 19E2, 099D,B4 ;20371          JMP [LOOP.T32.A]                 ;ERROR LOOP IF ENABLED
                ;20372
U 19E3, FF82,15 ;20373          INC [ERROR.NUMBER]               ;ERROR B
U 19E4, B700,15 ;20374          MOV [CKBTS.0111100] TO WR[0]     ;GET CKBTS FOR DATA OF 0 OR ALL 1'S
                ;20375                                           ; IN WR 0
U 19E5, A0C0,15 ;20376          COM WR[0]                        ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 19E6, 452C,15 ;20377          BIC LS[#FFFFFF00] TO WR[0]       ;CLEAR ALL BUT LOW BYTE
U 19E7, C776,15 ;20378          BIS LS[MME] TO WR[0]             ;SET MME BIT
U 19E8, 8A17,FC ;20379          JSR [WRITE.CSR1]                 ;PUT CKBTS FOR DATA OF 0 OR ALL 1'S AND
                ;20380                                           ; SET MME BIT IN CSR 1
U 19E9, B652,15 ;20381          MOV LS[#200] TO WR[0]            ;GET 200 IN WR
U 19EA, 2040,15 ;20382          INC WR[0]                        ;201 IN WR
U 19EB, BE12,15 ;20383          MOV WR[0] TO LS[T9]              ;ADDRESS TO WRITE IN LS (WITH LVA00 SET)
U 19EC, 5F60,15 ;20384          MCOM LS[#10000] TO WR[0]         ;FFFEFFFF IN WR 0
U 19ED, BE14,15 ;20385          MOV WR[0] TO LS[T10]             ;FFFEFFFF IN LS FOR DATA TO WRITE
                ;20386
U 19EE, 9D12,F5 ;20387          MEM.REQ[MAINT.ECC.DATA] ADRS[T9] DT[LONG]
                ;20388                                           ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;20389                                           ; SET FOR BRANCH
U 19EF, B214,15 ;20390          WRITE.MEM LS[T10]                ;WRITE A FFFEFFFF AT LOCATION 200 WITH
```

```
                ;20391                                          ; CKBTS FOR DATA OF ALL 1'S
                ;20392
U 19F0, 6C13,95 ;20393          ADD WR[3] TO LS[T9]              ;INCREMENT ADDRESS BY 4
U 19F1, 9D12,F5 ;20394          MEM.REQ[MAINT.ECC.DATA] ADRS[T9] DT[LONG]
                ;20395                                          ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;20396                                          ; SET FOR BRANCH
U 19F2, B260,15 ;20397          WRITE.MEM LS[#10000]            ;WRITE A 10000(H) AT LOCATION 204 WITH
                ;20398                                          ; CKBTS FOR DATA OF ALL 0'S
                ;20399
U 19F3, 6C13,95 ;20400          ADD WR[3] TO LS[T9]              ;INCREMENT ADDRESS BY 4
U 19F4, 9D12,F5 ;20401          MEM.REQ[MAINT.ECC.DATA] ADRS[T9] DT[LONG]
                ;20402                                          ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;20403                                          ; SET FOR BRANCH
U 19F5, B214,15 ;20404          WRITE.MEM LS[T10]               ;WRITE A FFFEFFFF AT LOCATION 208 WITH
                ;20405                                          ; CKBTS FOR DATA OF ALL 1'S
                ;20406
U 19F6, 6C13,95 ;20407          ADD WR[3] TO LS[T9]              ;INCREMENT ADDRESS BY 4
U 19F7, 9D12,F5 ;20408          MEM.REQ[MAINT.ECC.DATA] ADRS[T9] DT[LONG]
                ;20409                                          ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;20410                                          ; SET FOR BRANCH
U 19F8, B260,15 ;20411          WRITE.MEM LS[#10000]            ;WRITE A 10000(H) AT LOCATION 20C WITH
                ;20412                                          ; CKBTS FOR DATA OF ALL 0'S
                ;20413
U 19F9, 6C13,95 ;20414          ADD WR[3] TO LS[T9]              ;INCREMENT ADDRESS BY 4
U 19FA, 9D12,F5 ;20415          MEM.REQ[MAINT.ECC.DATA] ADRS[T9] DT[LONG]
                ;20416                                          ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;20417                                          ; SET FOR BRANCH
U 19FB, B214,15 ;20418          WRITE.MEM LS[T10]               ;WRITE A FFFEFFFF AT LOCATION 210 WITH
                ;20419                                          ; CKBTS FOR DATA OF ALL 1'S
                ;20420  LOOP.T32.B:
U 19FC, 9E43,75 ;20421          MEM.REQ[OCTA.READ.V.RCHK] ADRS[#2] DT[LONG]
                ;20422                                          ;SET UP TO EXECUTE UNALIGNED OCTA READ
U 19FD, 3022,15 ;20423          MOV MEM.DATA TO WR[0]           ;GET FIRST LONGWORD DATA
U 19FE, B814,15 ;20424          MOV MEM.DATA TO LS[T10]         ;GET SECOND LONGWORD DATA
U 19FF, 3816,15 ;20425          MOV MEM.DATA TO LS[T11]         ;GET THIRD LONGWORD DATA
U 1A00, B818,15 ;20426          MOV MEM.DATA TO LS[T12]         ;GET FOURTH LONGWORD DATA
                ;20427
U 1A01, 09A9,EC ;20428          JSR [CHECK.T32.OCTA]            ;CHECK RESULTS
U 1A02, 099F,C4 ;20429          JMP [LOOP.T32.B]                ;ERROR LOOP IF ENABLED
                ;20430
U 1A03, FF82,15 ;20431          INC LS[ERROR.NUMBER]            ;ERROR C
U 1A04, B652,15 ;20432          MOV LS[#200] TO WR[0]           ;GET 200 IN WR
U 1A05, 2040,15 ;20433          INC WR[0]                       ;201 IN WR
U 1A06, BE12,15 ;20434          MOV WR[0] TO LS[T9]              ;ADDRESS TO WRITE IN LS (WITH LVA00 SET)
U 1A07, 5F60,15 ;20435          MCOM LS[#10000] TO WR[0]        ;FFFEFFFF IN WR 0
U 1A08, 4562,15 ;20436          BIC LS[BIT17] TO WR[0]          ;FFFCFFFF IN WR 0
U 1A09, 3E1A,15 ;20437          MOV WR[0] TO LS[T13]            ;FFFCFFFF IN LS FOR DATA TO WRITE
                ;20438
U 1A0A, 9D12,F5 ;20439          MEM.REQ[MAINT.ECC.DATA] ADRS[T9] DT[LONG]
                ;20440                                          ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;20441                                          ; SET FOR BRANCH
U 1A0B, 321A,15 ;20442          WRITE.MEM LS[T13]               ;WRITE A FFFCFFFF AT LOCATION 200 WITH
                ;20443                                          ; CKBTS FOR DATA OF ALL 1'S
U 1A0C, DF40,15 ;20444          MCOM LS[#1] TO WR[0]            ;FFFFFFFE IN WR
U 1A0D, C542,15 ;20445          BIC LS[BIT1] TO WR[0]           ;FFFFFFFC IN WR
```

```
U 1A0E, 3E1A,15 ;20446          MOV WR[0] TO LS[T13]             ;EXPECTED DATA IN LS
U 1A0F, 3E89,15 ;20447          MOV WR[2] TO LS[ADDRESS.DATA]    ;ADDRESS FOR ERROR PRINTOUT (202)
                ;20448  LOOP.T32.C:
U 1A10, 1F43,75 ;20449          MEM.REQ[QUAD.READ.V.RCHK] ADRS[#2] DT[LONG]
                ;20450                                          ;SET UP TO EXECUTE UNALIGNED QUAD READ
U 1A11, 3022,15 ;20451          MOV MEM.DATA TO WR[0]           ;GET FIRST LONGWORD DATA (SHOULD ABORT
                ;20452                                          ; AFTER THIS)
U 1A12, B814,15 ;20453          MOV MEM.DATA TO LS[T10]         ;GET SECOND LONGWORD DATA
                ;20454
U 1A13, 361A,95 ;20455          MOV LS[T13] TO WR[1]            ;GET EXPECTED DATA (FFFFFFFC)
U 1A14, 0869,3C ;20456          JSR [CHECK.RESULT]              ;GET RESULT AND CHECK
U 1A15, 89A1,04 ;20457          JMP [LOOP.T32.C]                ;ERROR LOOP IF ENABLED
                ;20458
U 1A16, 1952,75 ;20459          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                ;20460                                          ;SET UP TO WRITE GOOD DATA IN 200
U 1A17, 329E,15 ;20461          WRITE.MEM LS[ONES]              ;WRITE ALL 1'S IN MEM
U 1A18, 6C13,95 ;20462          ADD WR[3] TO LS[T9]             ;INCREMENT ADDRESS BY 4
U 1A19, 9D12,F5 ;20463          MEM.REQ[MAINT.ECC.DATA] ADRS[T9] DT[LONG]
                ;20464                                          ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;20465                                          ; SET FOR BRANCH
U 1A1A, B2C0,15 ;20466          WRITE.MEM LS[#3(H)]             ;WRITE A 3(H) AT LOCATION 204 WITH
                ;20467                                          ; CKBTS FOR DATA OF ALL 0'S
                ;20468
U 1A1B, FF82,15 ;20469          INC LS[ERROR.NUMBER]            ;ERROR D
U 1A1C, 3628,15 ;20470          MOV LS[#FFFF] TO WR[0]          ;FFFF IN WR
U 1A1D, 4760,15 ;20471          BIS LS[BIT16] TO WR[0]          ;1FFFF IN WR
U 1A1E, C762,15 ;20472          BIS LS[BIT17] TO WR[0]          ;3FFFF IN WR
U 1A1F, 3E1C,15 ;20473          MOV WR[0] TO LS[T14]            ;EXPECTED DATA (3FFFF)
                ;20474  LOOP.T32.D:
U 1A20, 3E89,15 ;20475          MOV WR[2] TO LS[ADDRESS.DATA]   ;ADDRESS FOR ERROR PRINTOUT (202)
U 1A21, 1F43,75 ;20476          MEM.REQ[QUAD.READ.V.RCHK] ADRS[#2] DT[LONG]
                ;20477                                          ;SET UP TO EXECUTE UNALIGNED QUAD READ
U 1A22, 3022,15 ;20478          MOV MEM.DATA TO WR[0]           ;GET FIRST LONGWORD DATA
U 1A23, B814,15 ;20479          MOV MEM.DATA TO LS[T10]         ;GET SECOND LONGWORD DATA (SHOULD ABORT
                ;20480                                          ; AFTER THIS)
                ;20481
U 1A24, 361C,95 ;20482          MOV LS[T14] TO WR[1]            ;GET EXPECTED DATA (3FFFF)
U 1A25, 0869,3C ;20483          JSR [CHECK.RESULT]              ;GET RESULT AND CHECK
U 1A26, 89A2,04 ;20484          JMP [LOOP.T32.D]                ;ERROR LOOP IF ENABLED
                ;20485
U 1A27, DF28,95 ;20486          MCOM LS[#FFFF] TO WR[1]         ;EXPECTED DATA IF ABORT FAILS (ADDR 206)
U 1A28, 3614,15 ;20487          MOV LS[T10] TO WR[0]            ;GET SECOND LONGWORD
U 1A29, 89AB,2C ;20488          JSR [CHECK.T32.ABORT]           ;CHECK FOR ABORT
U 1A2A, 89A2,04 ;20489          JMP [LOOP.T32.D]                ;ERROR LOOP IF ENABLED
                ;20490
U 1A2B, FC12,15 ;20491          DEC LS[T9]                      ;204 NOW IN T9
U 1A2C, 9912,75 ;20492          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;20493                                          ;SET UP TO WRITE GOOD DATA IN 204
U 1A2D, B29C,15 ;20494          WRITE.MEM LS[ZERO]              ;WRITE ALL 0'S IN MEM
U 1A2E, FF12,15 ;20495          INC LS[T9]                      ;205 IN LS NOW
U 1A2F, 6C13,95 ;20496          ADD WR[3] TO LS[T9]             ;INCREMENT ADDRESS BY 4
U 1A30, 9D12,F5 ;20497          MEM.REQ[MAINT.ECC.DATA] ADRS[T9] DT[LONG]
                ;20498                                          ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;20499                                          ; SET FOR BRANCH
U 1A31, 321A,15 ;20500          WRITE.MEM LS[T13]               ;WRITE A FFFFFFFC AT LOCATION 208 WITH
```

```
                ;20501                                  ; CKBTS FOR DATA OF ALL 1'S
                ;20502
U 1A32, FF82,15 ;20503          INC LS[ERROR.NUMBER]    ;ERROR E
U 1A33, 5F28,15 ;20504          MCOM LS[#FFFF] TO WR[0] ;FFFF0000 IN WR
U 1A34, C560,15 ;20505          BIC LS[BIT16] TO WR[0]  ;FFFE0000 IN WR
U 1A35, 4562,15 ;20506          BIC LS[BIT17] TO WR[0]  ;FFFC0000 IN WR
U 1A36, 3E1A,15 ;20507          MOV WR[0] TO LS[T13]    ;EXPECTED DATA
                ;20508  LOOP.T32.E:
U 1A37, 3E89,15 ;20509          MOV WR[2] TO LS[ADDRESS.DATA]   ;202 IN ADDRESS FOR PRINTOUT
U 1A38, 9E43,75 ;20510          MEM.REQ[OCTA.READ.V.RCHK] ADRS[#2] DT[LONG]
                ;20511                                  ;SET UP TO EXECUTE UNALIGNED OCTA READ
U 1A39, 3022,15 ;20512          MOV MEM.DATA TO WR[0]   ;GET FIRST LONGWORD DATA
U 1A3A, B814,15 ;20513          MOV MEM.DATA TO LS[T10] ;GET SECOND LONGWORD DATA
U 1A3B, 3816,15 ;20514          MOV MEM.DATA TO LS[T11] ;GET THIRD LONGWORD DATA(SHOULD ABORT
                ;20515                                  ; AFTER THIS)
U 1A3C, B818,15 ;20516          MOV MEM.DATA TO LS[T12] ;GET FOURTH LONGWORD DATA
                ;20517
U 1A3D, B628,95 ;20518          MOV LS[#FFFF] TO WR[1]  ;GET EXPECTED DATA (0000FFFF)
U 1A3E, 0869,3C ;20519          JSR [CHECK.RESULT]      ;GET RESULT AND CHECK
U 1A3F, 89A3,74 ;20520          JMP [LOOP.T32.E]        ;ERROR LOOP IF ENABLED
                ;20521
U 1A40, 6C89,95 ;20522          ADD WR[3] TO LS[ADDRESS.DATA]   ;ADD 4 TO ADDRESS TO PRINT (206 NOW)
U 1A41, 361A,95 ;20523          MOV LS[T13] TO WR[1]    ;EXPECTED DATA (FFFC0000)
U 1A42, 3614,15 ;20524          MOV LS[T10] TO WR[0]    ;RECEIVED DATA
U 1A43, 0869,3C ;20525          JSR [CHECK.RESULT]      ;GET EXPECTED DATA AND CHECK
U 1A44, 89A3,74 ;20526          JMP [LOOP.T32.E]        ;ERROR LOOP IF ENABLED
                ;20527
U 1A45, B628,95 ;20528          MOV LS[#FFFF] TO WR[1]  ;EXPECTED DATA IF NO ABORT (ADDR 20A)
U 1A46, B616,15 ;20529          MOV LS[T11] TO WR[0]    ;RECEIVED DATA
U 1A47, 89AB,2C ;20530          JSR [CHECK.T32.ABORT]   ;CHECK FOR ABORT
U 1A48, 89A3,74 ;20531          JMP [LOOP.T32.E]        ;ERROR LOOP IF ENABLED
                ;20532
U 1A49, FF82,15 ;20533          INC LS[ERROR.NUMBER]    ;ERROR F
U 1A4A, B610,15 ;20534          MOV LS[T8] TO WR[0]     ;GET TB DATA
U 1A4B, 4540,15 ;20535          BIC LS[BIT0] TO WR[0]   ;MAP TO ADDRESS 0
U 1A4C, 3E10,15 ;20536          MOV WR[0] TO LS[T8]     ;PUT DATA IN LS
U 1A4D, 989C,F5 ;20537          MEM.REQ[WRITE.TB] ADRS[#0] DT[LONG]
                ;20538                                  ;SET UP TO WRITE TB ADDRESS 0
U 1A4E, 3210,15 ;20539          WRITE.MEM LS[T8]        ;SET VALID, PROT C, AND BYTE OFFSET
                ;20540                                  ; BITS AND MAP VIRTUAL ADDRESS 0 TO
                ;20541                                  ; PHYSICAL ADDRESS 0 IN TB
                ;20542
U 1A4F, B610,15 ;20543          MOV LS[T8] TO WR[0]     ;GET TB DATA
U 1A50, C57E,15 ;20544          BIC LS[BIT31] TO WR[0]  ;CLEAR VALID BIT
U 1A51, C740,15 ;20545          BIS LS[BIT0] TO WR[0]   ;MAP VIRTUAL 0 TO PHYSICAL 200
U 1A52, 3E10,15 ;20546          MOV WR[0] TO LS[T8]     ;PUT DATA IN LS
U 1A53, 1852,F5 ;20547          MEM.REQ[WRITE.TB] ADRS[BIT9] DT[LONG]
                ;20548                                  ;SET UP TO WRITE TB ADDRESS 1
U 1A54, 3210,15 ;20549          WRITE.MEM LS[T8]        ;SET  PROT C, AND BYTE OFFSET
                ;20550                                  ; BITS AND MAP VIRTUAL ADDRESS 0 TO
                ;20551                                  ; PHYSICAL ADDRESS 200 IN TB
                ;20552
U 1A55, B652,15 ;20553          MOV LS[#200] TO WR[0]   ;200 IN WR
U 1A56, 4144,15 ;20554          SUB LS[#4] FROM WR[0]   ;1FC IN WR
U 1A57, BE12,15 ;20555          MOV WR[0] TO LS[T9]     ;ADDRESS 1FC IN LS
```

```
U 1A58, 9912,75 ;20556          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;20557                                  ;SET UP TO WRITE IN ADDRESS 1FC
U 1A59, B29C,15 ;20558          WRITE.MEM LS[ZERO]      ;WRITE ZEROS AT 1FC
U 1A5A, C042,15 ;20559          ADD LS[#2] TO WR[0]     ;1FE IN WR
U 1A5B, BE12,15 ;20560          MOV WR[0] TO LS[T9]     ;ADDRESS TO READ (UNALIGNED READ)
U 1A5C, 2001,15 ;20561          MOV WR[0] TO WR[2]      ;SAVE ADDRESS IN WR 2
                ;20562  LOOP.T32.F:
U 1A5D, 3F89,15 ;20563          MOV WR[2] TO LS[ADDRESS.DATA]   ;1FE IN ADDRESS FOR PRINTOUT
U 1A5E, 1F13,75 ;20564          MEM.REQ[QUAD.READ.V.RCHK] ADRS[T9] DT[LONG]
                ;20565                                  ;SET UP TO EXECUTE UNALIGNED QUAD READ
                ;20566                                  ; AT ADDRESS 1FE
U 1A5F, 3022,15 ;20567          MOV MEM.DATA TO WR[0]   ;GET FIRST LONGWORD DATA
U 1A60, B814,15 ;20568          MOV MEM.DATA TO LS[T10] ;GET SECOND LONGWORD DATA (SHOULD ABORT
                ;20569                                  ; BEFORE THIS)
                ;20570
U 1A61, DF28,95 ;20571          MCOM LS[#FFFF] TO WR[1] ;GET EXPECTED DATA (FFFF0000)
U 1A62, 0869,3C ;20572          JSR [CHECK.RESULT]      ;GET RESULT AND CHECK
U 1A63, 89A5,D4 ;20573          JMP [LOOP.T32.F]        ;ERROR LOOP IF ENABLED
                ;20574
U 1A64, B628,95 ;20575          MOV LS[#FFFF] TO WR[1]  ;EXPECTED DATA IF NO ABORT (ADDR 202)
U 1A65, 3614,15 ;20576          MOV LS[T10] TO WR[0]    ;RECEIVED DATA
U 1A66, 89AB,2C ;20577          JSR [CHECK.T32.ABORT]   ;CHECK FOR ABORT
U 1A67, 89A5,D4 ;20578          JMP [LOOP.T32.F]        ;ERROR LOOP IF ENABLED
                ;20579
U 1A68, FF82,15 ;20580          INC LS[ERROR.NUMBER]    ;ERROR 10
U 1A69, B652,15 ;20581          MOV LS[#200] TO WR[0]   ;200 IN WR
U 1A6A, C146,15 ;20582          SUB LS[#8] FROM WR[0]   ;1F8 IN WR
U 1A6B, BE12,15 ;20583          MOV WR[0] TO LS[T9]     ;ADDRESS 1F8 IN LS
U 1A6C, 9912,75 ;20584          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;20585                                  ;SET UP TO WRITE IN ADDRESS 1F8
U 1A6D, 329E,15 ;20586          WRITE.MEM LS[ONES]      ;WRITE ONES AT 1F8
U 1A6E, C042,15 ;20587          ADD LS[#2] TO WR[0]     ;1FA IN WR
U 1A6F, BE12,15 ;20588          MOV WR[0] TO LS[T9]     ;ADDRESS TO READ (UNALIGNED READ)
U 1A70, 2001,15 ;20589          MOV WR[0] TO WR[2]      ;SAVE ADDRESS IN WR 2
                ;20590  LOOP.T32.10:
U 1A71, 3E89,15 ;20591          MOV WR[2] TO LS[ADDRESS.DATA]   ;1FA IN ADDRESS FOR PRINTOUT
U 1A72, 9E13,75 ;20592          MEM.REQ[OCTA.READ.V.RCHK] ADRS[T9] DT[LONG]
                ;20593                                  ;SET UP TO EXECUTE UNALIGNED OCTA READ
                ;20594                                  ; AT ADDRESS 1FA
U 1A73, 3022,15 ;20595          MOV MEM.DATA TO WR[0]   ;GET FIRST LONGWORD DATA
U 1A74, B814,15 ;20596          MOV MEM.DATA TO LS[T10] ;GET SECOND LONGWORD DATA (SHOULD ABORT
                ;20597                                  ; AFTER THIS)
U 1A75, 3816,15 ;20598          MOV MEM.DATA TO LS[T11] ;GET THIRD LONGWORD DATA
U 1A76, B818,15 ;20599          MOV MEM.DATA TO LS[T12] ;GET FOURTH LONGWORD DATA
                ;20600
U 1A77, B628,95 ;20601          MOV LS[#FFFF] TO WR[1]  ;GET EXPECTED DATA (0000FFFF)
U 1A78, 0869,3C ;20602          JSR [CHECK.RESULT]      ;GET RESULT AND CHECK
U 1A79, 09A7,14 ;20603          JMP [LOOP.T32.10]       ;ERROR LOOP IF ENABLED
                ;20604
U 1A7A, 6C89,95 ;20605          ADD WR[3] TO LS[ADDRESS.DATA]   ;ADD 4 TO ADDRESS TO PRINT OUT (1FE)
U 1A7B, DF28,95 ;20606          MCOM LS[#FFFF] TO WR[1] ;GET EXPECTED DATA (FFFF0000)
U 1A7C, 3614,15 ;20607          MOV LS[T10] TO WR[0]    ;RECEIVED DATA
U 1A7D, 0869,3C ;20608          JSR [CHECK.RESULT]      ;GET EXPECTED DATA AND CHECK
U 1A7E, 09A7,14 ;20609          JMP [LOOP.T32.10]       ;ERROR LOOP IF ENABLED
                ;20610
```

```
U 1A7F, B628,95 ;20611          MOV LS[#FFFF] TO WR[1]          ;EXPECTED DATA IF NO ABORT (ADDR 202)
U 1A80, B616,15 ;20612          MOV LS[T11] TO WR[0]            ;RECEIVED DATA
U 1A81, 89AB,2C ;20613          JSR [CHECK.T32.ABORT]           ;CHECK FOR ABORT
U 1A82, 09A7,14 ;20614          JMP [LOOP.T32.10]               ;ERROR LOOP IF ENABLED
                ;20615
U 1A83, FF82,15 ;20616          INC LS[ERROR.NUMBER]            ;ERROR 11
U 1A84, 1C52,75 ;20617          MEM.REQ[OCTA.WRITE.P] ADRS[#200] DT[LONG]
                ;20618                                          ;SET UP TO WRITE 4 LONGWORDS IN MEMORY
U 1A85, 329E,15 ;20619          WRITE.MEM LS[ONES]              ;WRITE ALL ONES PATTERN AT 200
U 1A86, B29C,15 ;20620          WRITE.MEM LS[ZERO]              ;WRITE ALL ZEROS PATTERN AT 204
U 1A87, 329E,15 ;20621          WRITE.MEM LS[ONES]              ;WRITE ALL ONES PATTERN AT 208
U 1A88, B29C,15 ;20622          WRITE.MEM LS[ZERO]              ;WRITE ALL ZEROS PATTERN AT 20C
                ;20623
U 1A89, E51C,15 ;20624          CLR LS[T14]                     ;0 IN LS
U 1A8A, FD1C,15 ;20625          COM LS[T14]                     ;ALL 1'S IN LS (EXPECTED DATA FOR FIRST
                ;20626                                          ; AND THIRD LONGWORD.  DATA FOR SECOND
                ;20627                                          ; AND FOURTH LONGWORD IS ALL 0'S)
U 1A8B, 3653,15 ;20628          MOV LS[#200] TO WR[2]           ;STARTING ADDRESS
                ;20629  LOOP.T32.11:
U 1A8C, 9E52,75 ;20630          MEM.REQ[OCTA.READ.P] ADRS[#200] DT[LONG]
                ;20631                                          ;SET UP TO EXECUTE OCTA PHYSICAL READ
U 1A8D, 3022,15 ;20632          MOV MEM.DATA TO WR[0]           ;GET FIRST LONGWORD DATA
U 1A8E, B814,15 ;20633          MOV MEM.DATA TO LS[T10]         ;GET SECOND LONGWORD DATA
U 1A8F, 3816,15 ;20634          MOV MEM.DATA TO LS[T11]         ;GET THIRD LONGWORD DATA
U 1A90, B818,15 ;20635          MOV MEM.DATA TO LS[T12]         ;GET FOURTH LONGWORD DATA
                ;20636
U 1A91, 09A9,EC ;20637          JSR [CHECK.T32.OCTA]            ;CHECK RESULTS
U 1A92, 89A8,C4 ;20638          JMP [LOOP.T32.11]               ;ERROR LOOP IF ENABLED
                ;20639
U 1A93, 09AB,E4 ;20640          JMP [END.T32]                   ;DONE
                ;20641
                ;20642  ;
                ;20643  ; THIS SUBROUTINE CHECKS 2 LONGWORDS OF DATA RETURNED BY QUAD READ.  EXPECTED
                ;20644  ; RESULT FOR FIRST LONGWORD IS IN LS LOCATION T14, AND FOR SECOND LONGWORD IN
                ;20645  ; LS T15
                ;20646  ;
                ;20647
                ;20648  CHECK.T32.QUAD:
U 1A94, 3E89,15 ;20649          MOV WR[2] TO LS[ADDRESS.DATA]   ;ADDRESS FOR PRINTOUT
U 1A95, 361C,95 ;20650          MOV LS[T14] TO WR[1]            ;GET EXPECTED DATA
U 1A96, 0869,3C ;20651          JSR [CHECK.RESULT]              ;CHECK FOR CORRECT DATA
U 1A97, 5B00,14 ;20652          RETURN                          ;ERROR LOOP IF ENABLED
                ;20653
U 1A98, BE89,95 ;20654          MOV WR[3] TO LS[ADDRESS.DATA]   ;ADDRESS FOR PRINTOUT
U 1A99, B61E,95 ;20655          MOV LS[T15] TO WR[1]            ;EXPECTED DATA FOR SECOND LONGWORD
U 1A9A, 3614,15 ;20656          MOV LS[T10] TO WR[0]            ;GET RESULT FROM LS
U 1A9B, 0869,3C ;20657          JSR [CHECK.RESULT]              ;CHECK FOR CORRECT DATA
U 1A9C, 5B00,14 ;20658          RETURN                          ;ERROR LOOP IF ENABLED
                ;20659
U 1A9D, DB00,16 ;20660          RETURN+1                        ;NO ERROR LOOP RETURN
                ;20661
                ;20662  ;
                ;20663  ; THIS SUBROUTINE CHECKS 4 LONGWORDS OF DATA RETURNED BY OCTA READ.  EXPECTED
                ;20664  ; RESULT (FOR FIRST LONGWORD) IS IN LS LOCATION T14.
                ;20665  ;
```

```
                    ;20666
                    ;20667  CHECK.T32.OCTA:
U 1A9E, 3E89,15     ;20668          MOV WR[2] TO LS[ADDRESS.DATA]   ;ADDRESS FOR ERROR PRINTOUT
U 1A9F, 361C,95     ;20669          MOV LS[T14] TO WR[1]            ;GET EXPECTED DATA
U 1AA0, 0869,3C     ;20670          JSR [CHECK.RESULT]              ;GET RESULT AND CHECK
U 1AA1, 5B00,14     ;20671          RETURN                          ;ERROR LOOP IF ENABLED
                    ;20672
U 1AA2, 6C89,95     ;20673          ADD WR[3] TO LS[ADDRESS.DATA]   ;ADDRESS FOR ERROR PRINTOUT
U 1AA3, 5F1C,95     ;20674          MCOM LS[T14] TO WR[1]           ;GET EXPECTED DATA
U 1AA4, 3614,15     ;20675          MOV LS[T10] TO WR[0]            ;RECEIVED DATA
U 1AA5, 0869,3C     ;20676          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 1AA6, 5B00,14     ;20677          RETURN                          ;ERROR LOOP IF ENABLED
                    ;20678
U 1AA7, 6C89,95     ;20679          ADD WR[3] TO LS[ADDRESS.DATA]   ;ADDRESS FOR ERROR PRINTOUT
U 1AA8, 361C,95     ;20680          MOV LS[T14] TO WR[1]            ;GET EXPECTED DATA
U 1AA9, B616,15     ;20681          MOV LS[T11] TO WR[0]            ;RECEIVED DATA
U 1AAA, 0869,3C     ;20682          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 1AAB, 5B00,14     ;20683          RETURN                          ;ERROR LOOP IF ENABLED
                    ;20684
U 1AAC, 6C89,95     ;20685          ADD WR[3] TO LS[ADDRESS.DATA]   ;ADDRESS FOR ERROR PRINTOUT
U 1AAD, 5F1C,95     ;20686          MCOM LS[T14] TO WR[1]           ;GET EXPECTED DATA
U 1AAE, 3618,15     ;20687          MOV LS[T12] TO WR[0]            ;RECEIVED DATA
U 1AAF, 0869,3C     ;20688          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 1AB0, 5B00,14     ;20689          RETURN                          ;ERROR LOOP IF ENABLED
                    ;20690
U 1AB1, DB00,16     ;20691          RETURN+1                        ;NO ERROR LOOP RETURN
                    ;20692
                    ;20693  ;
                    ;20694  ;  This routine checks that an abort occured by comparing the received data
                    ;20695  ;  with the data that would come back if no abort occurred.  If the data is
                    ;20696  ;  equal, an error is faked and reported.
                    ;20697  ;
                    ;20698
                    ;20699  CHECK.T32.ABORT:
U 1AB2, 6C89,95     ;20700          ADD WR[3] TO LS[ADDRESS.DATA]   ;INCREMENT ADDRESS TO PRINT BY 4
                    ;20701          CMP WR[0] WITH WR[1],           ;SEE IF RECEIVED DATA EQUALS WHAT WOULD
                    ;20702                                          ; COME BACK IF NO ABORT
U 1AB3, 2640,B5     ;20703           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1AB4, 89AB,91     ;20704          JMP.IF[NEQ] TO [T32.RET+1]      ;JMP IF NOT, DATA OK
                    ;20705
U 1AB5, B66E,15     ;20706          MOV LS[NA.EXP.REC] TO WR[0]     ;SET BIT TO INHIBIT EXP AND REC PRINTOUT
U 1AB6, 6D80,15     ;20707          BIS WR[0] TO LS[CONTROL.STATUS] ;SET BIT IN CONTROL AND STATUS WORD
U 1AB7, 0869,3C     ;20708          JSR [CHECK.RESULT]              ;CALL ERROR ROUTINE WITH DATA SET UP TO
                    ;20709                                          ; FAIL
U 1AB8, 09AB,B4     ;20710          JMP [CLEAR.T32.NA]              ;CLEAR INHIBIT EXP AND REC
                    ;20711  T32.RET+1:
U 1AB9, 89AB,BC     ;20712          JSR [CLEAR.T32.NA]              ;CLEAR INHIBIT BIT IF SET
U 1ABA, DB00,16     ;20713          RETURN+1                        ;NO LOOP RETURN
                    ;20714
                    ;20715  CLEAR.T32.NA:
U 1ABB, B670,15     ;20716          MOV LS[OTHER.DATA] TO WR[0]     ;SET BIT TO PRINT UNDER OTHER IN ERROR
                    ;20717                                          ; REPORT.  CLEAR INHIBIT BIT
U 1ABC, 3E80,15     ;20718          MOV WR[0] TO LS[CONTROL.STATUS] ;PUT IN CONTROL AND STATUS WORD
                    ;20719
U 1ABD, 5B00,14     ;20720          RETURN                          ;ERROR LOOP RETURN OR RETURN FOR JSR
```

```
;20721                                          ; AT T32.RET+1
;20722
;20723  END.T32:
```

```
;20724  .page   '' Extensive Array and Memory Timing Test''
;20725  .toc    ''  TEST 33 Moving Inversion Test on Array (M8728  ARRAY or M8391 MCT Module )''
;20726  ;++
;20727  ;
;20728  ; Test Description:
;20729  ;
;20730  ;       This test runs a moving inversion test on main memory.  The moving in-
;20731  ;       version test is similar to a march test except for the way that the
;20732  ;       address is incremented.  The address is incremented in such a way that
;20733  ;       each address bit in turn is toggled at the maximum rate.  This is ac-
;20734  ;       complished by adding a shifting 1 constant at each new pass through
;20735  ;       the test.  The first pass through the test will write a background of
;20736  ;       0's, then read 0's, write 1's at address 0.  The constant 4 is then
;20737  ;       added to the address and the read/write is repeated.  This continues
;20738  ;       until all memory is accessed.  The array is then checked using a de-
;20739  ;       scending address in the same manner.  This toggles bit 2 at the max-
;20740  ;       imum rate.
;20741  ;         The second pass is similar to the first except that a 8 is used as
;20742  ;       the constant to add to the address.  When the top of memory is reached,
;20743  ;       bit 2 is incremented and the increment repeated.  The addressing will
;20744  ;       look like this (in hex): 0 8 10 18 ... to top of memory then 4 C 14 1C
;20745  ;       ... to top of memory again.  Thus bit 3 is toggled at the maximum rate.
;20746  ;       Then the test is repeated using byte writes and reads while toggling
;20747  ;       bits 0 and 1 at their maximum rate.
;20748  ;         Both tests are repeated with complemented data on the next pass.
;20749  ;
;20750  ;         A march test using unaligned longword accesses and octaword accesses
;20751  ;       is run to pick up timing related problems in these flows.  These are
;20752  ;       run without toggeling different bits to keep the execution time
;20753  ;       reasonable.  The octa word accesses are done starting at address 4 to
;20754  ;       maximize crossing page boundaries.
;20755  ;
;20756  ; Assumptions:
;20757  ;
;20758  ;       It is assumed that all previous tests have run succesfully.
;20759  ;
;20760  ; Test Steps:
;20761  ;
;20762  ;       1)  Set up error number and module number in LS (for error printouts)
;20763  ;           and clear previous error number in LS.
;20764  ;       2)  Size and initialize memory.  Report number of 256KB blocks found.
;20765  ;       3)  Write a background of ones in all available memory.
;20766  ;       4)  Starting at address 0, read 1,s, check result, then write 0's.  In-
;20767  ;           crement address by X (X=bit currently being toggled at maximum rate).
;20768  ;       5)  Repeat until top of memory is reached.  Then increment lower 18 bits
;20769  ;           by 4 and clear bits above bit 17.  If bit being toggled is now set,
;20770  ;           go on.  Else repeat steps 4 and 5 starting at current address.
;20771  ;       6)  Repeat steps 4 and 5 until each bit from bit 2 through bit 17 has
;20772  ;           been toggled at its maximum rate (i.e. X has been equal to each bit
;20773  ;           from 2 through 17).
;20774  ;       7)  Repeat steps 4 through 6 starting at top of memory and working down.
;20775  ;       8)  Repeat steps 4 through 8 using bits 0 and 1 as the toggled bit and
;20776  ;           doing byte writes and reads.
;20777  ;       9)  Repeat steps 4 through 8 with complement data.
;20778  ;       10) Run a normal march pattern using unaligned longword accesses. Run
```

```
                ;20779  ;           one pass up and one pass down.
                ;20780  ;       11) Same as step 10 with OCTA writes.
                ;20781  ;
                ;20782  ; Errors:
                ;20783  ;
                ;20784  ;       Note:  If a failure occurs, the address being accessed will be in LS 9
                ;20785  ;              and can be examined via EX LS 9.  The bit being toggled at the
                ;20786  ;              maximum rate will be in WR 3 and can be examined by typing EX
                ;20787  ;              WR 3.  OTHER data is not used so as to insure that the test runs
                ;20788  ;              as fast as possible.
                ;20789  ;
                ;20790  ;       Error 1 - Moving inversion failed on ascending path with background of
                ;20791  ;                 1's.  Bit being toggled is from bit 2 up through bit 17.
                ;20792  ;       Error 2 - Moving inversion failed on descending path with background of
                ;20793  ;                 1's.  Bit being toggled is from bit 2 up through bit 17.
                ;20794  ;       Error 3 - Moving inversion failed on ascending path with background of
                ;20795  ;                 1's.  Bit being toggled is either bit 0 or bit 1.
                ;20796  ;       Error 4 - Moving inversion failed on descending path with background of
                ;20797  ;                 1's.  Bit being toggled is either bit 0 or bit 1.
                ;20798  ;       Error 5 - Moving inversion failed on ascending path with background of
                ;20799  ;                 0's.  Bit being toggled is from bit 2 up through bit 17.
                ;20800  ;       Error 6 - Moving inversion failed on descending path with background of
                ;20801  ;                 0's.  Bit being toggled is from bit 2 up through bit 17.
                ;20802  ;       Error 7 - Moving inversion failed on ascending path with background of
                ;20803  ;                 0's.  Bit being toggled is either bit 0 or bit 1.
                ;20804  ;       Error 8 - Moving inversion failed on descending path with background of
                ;20805  ;                 0's.  Bit being toggled is either bit 0 or bit 1.
                ;20806  ;       Error 9 - March test on unaligned longwords failed on ascending path
                ;20807  ;                 with background of 0's.
                ;20808  ;       Error A - March test on unaligned longwords failed on descending path
                ;20809  ;                 with background of 0's.
                ;20810  ;       Error B - March test on octa accesses failed on ascending path with
                ;20811  ;                 background of 0's.
                ;20812  ;       Error C - March test on octa accesses failed on descending path with
                ;20813  ;                 background of 0's.
                ;20814  ;
                ;20815  ; Debug:
                ;20816  ;
                ;20817  ;       NOTE:  If the MCT is single stepped, refresh on the ARRAY module will
                ;20818  ;       be lost.  Thus the array data will be lost.
                ;20819  ;
                ;20820  ;       Error 1 through C - Any of these errors represents a possible problem
                ;20821  ;       with the array card being accessed as indicated by the address in LS 9.
                ;20822  ;       The address and data bit that fails can help pinpoint the failing RAM
                ;20823  ;       chip.  If all bits are failing, the problem could lie on the MCT board
                ;20824  ;       with a slow chip in the VAR logic.  Since this test is a check for
                ;20825  ;       logic operating correctly at speed, it is hard to pinpoint the problem
                ;20826  ;       without actually scopeing the address lines.
                ;20827  ;
                ;20828  ;--
                ;20829
                ;20830  T.33:
                ;20831
U 1ABE, B65E,15 ;20832          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;20833                                          ; STATUS WORD
```

```
U 1ABF, 3E80,15 ;20834          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;20835                                         ; BIT 15 INDICATES START OF TEST TO
                ;20836                                         ; CONSOLE
U 1AC0, 10E0,15 ;20837          MISC [SET.CP.ATTN]             ;ISSUE CPU ATTN TO CONSOLE
                ;20838  WAIT.T33.0:
U 1AC1, 89AC,14 ;20839          JMP [WAIT.T33.0]               ;LOOP HERE WAITING FOR CONSOLE TO
                ;20840                                         ; RESPOND
                ;20841
U 1AC2, B690,95 ;20842          MOV LS[ERROR.CONTROL] TO WR[1]  ;GET ERROR CONTROL WORD
                ;20843          BIT LS[APT.PRESENT] WITH WR[1], ;SEE IF UNDER APT
U 1AC3, D94A,B5 ;20844           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1AC4, 09AC,89 ;20845          JMP.IF[BITS.CLR] TO [T33.START] ;IF NOT APT, GO DO THIS TEST
                ;20846
U 1AC5, 3692,95 ;20847          MOV LS[APT.PARAM] TO WR[1]      ;ELSE GET APT PARAMETERS
                ;20848          BIT LS[#10000] WITH WR[1],      ;SEE IF SOFTWARE BIT SET UNDER APT
U 1AC6, 5960,B5 ;20849           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1AC7, 89BC,C1 ;20850          JMP.IF[BIT.SET] TO [END.T33]    ;IF BIT SET, SKIP TEST
                ;20851                                         ; ELSE DO THIS TEST
                ;20852
                ;20853  T33.START:
U 1AC8, 0A1A,9C ;20854          JSR [SETUP]                     ;SET UP MASKS, MODULE CODE ETC. AND
                ;20855                                         ; CLEAR MCT CSR 1
U 1AC9, 4748,15 ;20856          BIS LS[ARRAY] TO WR[0]          ;ADD ARRAY BOARD AS FAILING MODULE
U 1ACA, 3E8C,15 ;20857          MOV WR[0] TO LS[MODULE.NUM]     ;STORE IN LS (ARRAY OR MCT IS FAILING
                ;20858                                         ; MODULE)
                ;20859
                ;20860          MOV LS[MM.LASTADDR+1] TO WR[2], ;GET LAST ADDRESS + 1 FROM LS
U 1ACB, 3625,35 ;20861           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1ACC, 89AC,E1 ;20862          JMP.IF[NEQ] TO [T33.CONTINUE]   ;CONTINUE TESTING IF LAST ADDRESS+1 IS
                ;20863                                         ; NON-ZERO
U 1ACD, 8A19,4C ;20864          JSR [SIZE.MEMORY]               ;ELSE SIZE MEMORY, PRINT RESULT, AND
                ;20865                                         ; CONTINUE
                ;20866
                ;20867  T33.CONTINUE:
U 1ACE, 3628,15 ;20868          MOV LS[#FFFF] TO WR[0]          ;FFFF IN WR
U 1ACF, 4760,15 ;20869          BIS LS[#10000] TO WR[0]         ;1FFFF IN WR
U 1AD0, C762,15 ;20870          BIS LS[#20000] TO WR[0]         ;3FFFF IN WR
U 1AD1, 3E10,15 ;20871          MOV WR[0] TO LS[T8]             ;SAVE IN LS TO MASK LOWER BITS
                ;20872
U 1AD2, 6518,15 ;20873          CLR LS[T12]                     ;0 IN LS
U 1AD3, 7D18,15 ;20874          COM LS[T12]                     ;DATA OF ALL 1'S IN LS T12
U 1AD4, E51A,15 ;20875          CLR LS[T13]                     ;DATA OF ALL 0'S IN LS T13
U 1AD5, 09BC,0C ;20876          JSR [BACK.T33]                  ;GO WRITE BACKGROUND OF 1'S
U 1AD6, 89AE,BC ;20877          JSR [T33.1.5]                   ;GO DO TEST TOGGLING BITS 2 AND UP
                ;20878
U 1AD7, B62C,15 ;20879          MOV LS[#FFFFFF00] TO WR[0]      ;FFFFFF00 IN WR
U 1AD8, 3E8A,15 ;20880          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK 1 BYTE ONLY
U 1AD9, 09BC,9C ;20881          JSR [T33.INC.ERR]               ;SET ERROR NUMBER TO 3
U 1ADA, 89B1,CC ;20882          JSR [T33.3.7]                   ;GO DO TEST WITH BITS 0 AND 1 TOGGLING
                ;20883
U 1ADB, E58A,15 ;20884          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
U 1ADC, 09BC,9C ;20885          JSR [T33.INC.ERR]               ;SET ERROR NUMBER TO 5
U 1ADD, 7D18,15 ;20886          COM LS[T12]                     ;COMPLEMENT ALL 1'S DATA TO ALL 0'S DATA
U 1ADE, FD1A,15 ;20887          COM LS[T13]                     ;COMPLEMENT ALL 0'S DATA TO ALL 1'S DATA
U 1ADF, 09BC,0C ;20888          JSR [BACK.T33]                  ;WRITE BACKGROUND OF 0'S
```

```
U 1AE0, 89AE,BC ;20889          JSR [T33.1.5]                   ;DO TEST WITH COMPLEMENT DATA TOGGLING
                ;20890                                          ; BITS 2 AND UP
                ;20891
U 1AE1, B62C,15 ;20892          MOV LS[#FFFFFF00] TO WR[0]      ;FFFFFF00 IN WR
U 1AE2, 3E8A,15 ;20893          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK 1 BYTE ONLY
U 1AE3, 09BC,9C ;20894          JSR [T33.INC.ERR]               ;SET ERROR NUMBER TO 7
U 1AE4, 89B1,CC ;20895          JSR [T33.3.7]                   ;DO TEST WITH COMPLEMENT DATA AND BITS
                ;20896                                          ; 0 AND 1 TOGGLING
                ;20897
U 1AE5, 09BC,9C ;20898          JSR [T33.INC.ERR]               ;SET ERROR NUMBER TO 9
U 1AE6, E58A,15 ;20899          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
U 1AE7, 89B5,2C ;20900          JSR [T33.9]                     ;DO TEST WITH UNALIGNED LONGWORD ACCESSES
                ;20901
U 1AE8, FF82,15 ;20902          INC LS[ERROR.NUMBER]            ;SET ERROR NUMBER TO B
U 1AE9, 09B7,4C ;20903          JSR [T33.B]                     ;DO TEST WITH OCTA WORD ACCESSES
                ;20904
U 1AEA, 89BC,C4 ;20905          JMP [END.T33]                   ;DONE
                ;20906
                ;20907  T33.1.5:
U 1AEB, 3645,95 ;20908          MOV LS[BIT2] TO WR[3]           ;BIT WE ARE GOING TO TOGGLE AT MAX RATE
                ;20909                                          ; FIRST BIT TO TOGGLE IS RAM BIT 2
U 1AEC, 89BB,CC ;20910          JSR [CALL.MONITOR]              ;CALL MONITOR TO LET HIM KNOW WE'RE
                ;20911                                          ; STILL ALIVE
                ;20912  REPEAT.T33.1.5:
U 1AED, 6512,15 ;20913          CLR LS[T9]                      ;START AT ADDRESS 0 IN ARRAY
                ;20914  LOOP.T33.1.5:
U 1AEE, B618,95 ;20915          MOV LS[T12] TO WR[1]            ;EXPECTED DATA
U 1AEF, 1813,F5 ;20916          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;20917                                          ;SET UP TO READ ARRAY
U 1AF0, 3022,15 ;20918          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1AF1, 0869,3C ;20919          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 1AF2, 09AE,E4 ;20920          JMP [LOOP.T33.1.5]              ;ERROR LOOP IF ENABLED
                ;20921
U 1AF3, 9912,75 ;20922          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;20923                                          ;SET UP TO WRITE COMPLEMENT IN ARRAY
U 1AF4, 321A,15 ;20924          WRITE.MEM LS[T13]               ;WRITE COMP DATA IN THIS LOCATION
                ;20925
U 1AF5, 6C13,95 ;20926          ADD WR[3] TO LS[T9]             ;TOGGLE BIT IN ADDRESS
U 1AF6, 3612,15 ;20927          MOV LS[T9] TO WR[0]             ;GET UPDATED ADDRESS
U 1AF7, 4510,15 ;20928          BIC LS[T8] TO WR[0]             ;CLEAR LOWER 18 BITS
                ;20929          CMP WR[0] WITH LS[MM.LASTADDR+1], ;SEE IF INCREMENTED OVER TOP
U 1AF8, 4F24,35 ;20930           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1AF9, 09AE,E1 ;20931          JMP.IF[NEQ] TO [LOOP.T33.1.5]   ;IF NOT OVER TOP OF MEMORY, REPEAT AT
                ;20932                                          ; NEXT ADDRESS
U 1AFA, EC13,15 ;20933          ADD WR[2] TO LS[T9]             ;ELSE INCREMENT ADDRESS BELOW TOGGLED
                ;20934                                          ; BIT
U 1AFB, B610,15 ;20935          MOV LS[T8] TO WR[0]             ;GOING TO CLEAR BITS ABOVE BIT 17
U 1AFC, EE12,15 ;20936          AND WR[0] TO LS[T9]             ;CLEAR BITS ABOVE BIT 17
                ;20937          BIT WR[3] WITH LS[T9],          ;IF TOGGLED BIT IS NOW SET, WE'RE DONE
U 1AFD, D913,B5 ;20938           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 1AFE, 89AE,E9 ;20939          JMP.IF[BITS.CLR] TO [LOOP.T33.1.5] ;REPEAT TEST AT NEW ADDRESS IF NOT
                ;20940                                          ; DONE
                ;20941
                ;20942                                          ; ELSE GOING TO START DESCENDING ADDRS
U 1AFF, 89BB,CC ;20943          JSR [CALL.MONITOR]              ;CALL MONITOR TO LET HIM KNOW WE'RE
```

```
                    ;20944                                          ; STILL ALIVE
                    ;20945
U 1B00, FF82,15     ;20946          INC LS[ERROR.NUMBER]            ;NEXT ERROR NUMBER
U 1B01, 3624,15     ;20947          MOV LS[MM.LASTADDR+1] TO WR[0]  ;START AT TOP OF MEMORY
U 1B02, 4144,15     ;20948          SUB LS[#4] FROM WR[0]           ;DECREMENT BY 4 TO GET TOP VALID ADDR
U 1B03, BE14,15     ;20949          MOV WR[0] TO LS[T10]            ;SAVE IN LS
U 1B04, BE12,15     ;20950          MOV WR[0] TO LS[T9]             ;CURRENT ADDRESS IN T9
U 1B05, AE46,15     ;20951          SUB WR[3] FROM WR[0]            ;ALSO CALCULATE ADDRESS TO STOP AT
U 1B06, 3E16,15     ;20952          MOV WR[0] TO LS[T11]            ;SAVE THIS IN LS
                    ;20953  LOOP.T33.2.6:
U 1B07, 361A,95     ;20954          MOV LS[T13] TO WR[1]            ;EXPECTED DATA
U 1B08, 1813,F5     ;20955          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                    ;20956                                          ;SET UP TO READ ARRAY
U 1B09, 3022,15     ;20957          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1B0A, 0869,3C     ;20958          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 1B0B, C9B0,74     ;20959          JMP [LOOP.T33.2.6]              ;ERROR LOOP IF ENABLED
                    ;20960
U 1B0C, 9912,75     ;20961          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                    ;20962                                          ;SET UP TO WRITE COMPLEMENT IN ARRAY
U 1B0D, B218,15     ;20963          WRITE.MEM LS[T12]               ;WRITE COMP DATA IN THIS LS LOCATION
                    ;20964
                    ;20965          SUB WR[3] FROM LS[T9],          ;TOGGLE BIT IN ADDRESS
U 1B0E, 6B13,B5     ;20966           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1B0F, 89B0,70     ;20967          JMP.IF[N.CLR] TO [LOOP.T33.2.6] ;IF NOT LESS THAN ADDRESS 0, REPEAT AT
                    ;20968                                          ; NEXT ADDRESS
U 1B10, 3614,15     ;20969          MOV LS[T10] TO WR[0]            ;ELSE GET LAST ADDRESS WE STARTED AT
U 1B11, 2E44,15     ;20970          SUB WR[2] FROM WR[0]            ;DECREMENT ADDRESS
U 1B12, BE14,15     ;20971          MOV WR[0] TO LS[T10]            ;SAVE ADDRESS
U 1B13, BE12,15     ;20972          MOV WR[0] TO LS[T9]             ;ADDRESS IN T9
                    ;20973
                    ;20974          CMP LS[T11] WITH WR[0],         ;IF ADDRESS = T11, WE'RE DONE
U 1B14, 4E16,35     ;20975           DT(LONG)&SET.ALU.CC            ; SET CONDITION CODES
U 1B15, 09B0,71     ;20976          JMP.IF[NEQ] TO [LOOP.T33.2.6]   ;REPEAT TEST AT NEW ADDRESS IF NOT
                    ;20977                                          ; DONE
                    ;20978
U 1B16, 89BB,CC     ;20979          JSR [CALL.MONITOR]              ;CALL MONITOR TO LET HIM KNOW WE'RE
                    ;20980                                          ; STILL ALIVE
                    ;20981
U 1B17, FC82,15     ;20982          DEC LS[ERROR.NUMBER]            ;DECREMENT ERROR NUMBER
U 1B18, A3C1,95     ;20983          ROL WR[3]                       ;NEXT BIT TO TOGGLE AT MAX RATE
                    ;20984          BIT LS[#40000] WITH WR[3],      ;SEE IF BIT 18 HAS SET
U 1B19, 5965,B5     ;20985           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1B1A, 89AE,D9     ;20986          JMP.IF[BITS.CLR] TO [REPEAT.T33.1.5] ;REPEAT TEST WITH NEW TOGGLED BIT
                    ;20987                                          ; IF BIT 18 HAS NOT SET
U 1B1B, 5B00,14     ;20988          RETURN                          ;ELSE DONE
                    ;20989
                    ;20990
                    ;20991  T33.3.7:
U 1B1C, 3641,15     ;20992          MOV LS[BIT0] TO WR[2]           ;SET UP INCREMENTER (INCREMENTS ADDRESS
                    ;20993                                          ; BIT 0 AT RAM CHIP)
U 1B1D, B641,95     ;20994          MOV LS[BIT0] TO WR[3]           ;BIT WE ARE GOING TO TOGGLE AT MAX RATE
                    ;20995                                          ; FIRST BIT TO TOGGLE IS RAM BIT 0
U 1B1E, 89BB,AC     ;20996          JSR [SETUP.T33.COUNTER&CALL.MONITOR] ;SET UP COUNTER AND CALL MONITOR
                    ;20997                                          ; TO LET HIM KNOW WE'RE STILL ALIVE
                    ;20998
```

```
                    ;20999  REPEAT.T33.3.7:
U 1B1F, 6512,15     ;21000          CLR LS[T9]                      ;START AT ADDRESS 0 IN ARRAY
                    ;21001  LOOP.T33.3.7:
U 1B20, B618,95     ;21002          MOV LS[T12] TO WR[1]             ;EXPECTED DATA
U 1B21, 1813,95     ;21003          MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
                    ;21004                                          ;SET UP TO READ ARRAY
U 1B22, 3022,15     ;21005          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 1B23, 0869,3C     ;21006          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 1B24, 09B2,04     ;21007          JMP [LOOP.T33.3.7]               ;ERROR LOOP IF ENABLED
                    ;21008
U 1B25, 9912,15     ;21009          MEM.REQ[WRITE.P] ADRS[T9] DT[BYTE]
                    ;21010                                          ;SET UP TO WRITE COMPLEMENT IN ARRAY
U 1B26, 321A,15     ;21011          WRITE.MEM LS[T13]                ;WRITE COMP DATA IN THIS LOCATION
                    ;21012
                    ;21013          DEC LS[T14],                     ;DECREMENT COUNTER
U 1B27, FC1C,35     ;21014           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 1B28, 09B2,A1     ;21015          JMP.IF[NEQ] TO [CONT.T33.3.7]    ;JMP IF COUNTER NOT YET ZERO
                    ;21016
U 1B29, 89BB,AC     ;21017          JSR [SETUP.T33.COUNTER&CALL.MONITOR] ;SET UP COUNTER AND CALL MONITOR
                    ;21018                                          ; TO LET HIM KNOW WE'RE STILL ALIVE
                    ;21019  CONT.T33.3.7:
U 1B2A, 6C13,95     ;21020          ADD WR[3] TO LS[T9]              ;TOGGLE BIT IN ADDRESS
U 1B2B, 3612,15     ;21021          MOV LS[T9] TO WR[0]              ;GET UPDATED ADDRESS
U 1B2C, 4510,15     ;21022          BIC LS[T8] TO WR[0]              ;CLEAR LOWER 18 BITS
                    ;21023          CMP WR[0] WITH LS[MM.LASTADDR+1], ;SEE IF INCREMENTED OVER TOP
U 1B2D, 4F24,35     ;21024           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 1B2E, 09B2,01     ;21025          JMP.IF[NEQ] TO [LOOP.T33.3.7]    ;IF NOT OVER TOP OF MEMORY, REPEAT AT
                    ;21026                                          ; NEXT ADDRESS
U 1B2F, EC13,15     ;21027          ADD WR[2] TO LS[T9]              ;ELSE INCREMENT ADDRESS BELOW TOGGLED
                    ;21028                                          ; BIT
U 1B30, B610,15     ;21029          MOV LS[T8] TO WR[0]              ;GOING TO CLEAR BITS ABOVE BIT 17
U 1B31, EE12,15     ;21030          AND WR[0] TO LS[T9]              ;CLEAR BITS ABOVE BIT 17
                    ;21031          BIT WR[3] WITH LS[T9],           ;IF TOGGLED BIT IS NOW SET, WE'RE DONE
U 1B32, D913,B5     ;21032           DT(LONG)&SET.ALU.CC             ; SET CONDITION CODES
U 1B33, 89B2,09     ;21033          JMP.IF[BITS.CLR] TO [LOOP.T33.3.7] ;REPEAT TEST AT NEW ADDRESS IF NOT
                    ;21034                                          ; DONE
                    ;21035
                    ;21036                                          ; ELSE GOING TO START DESCENDING ADDRS
U 1B34, FF82,15     ;21037          INC LS[ERROR.NUMBER]             ;ERROR 6 OR 8
U 1B35, 3624,15     ;21038          MOV LS[MM.LASTADDR+1] TO WR[0]   ;START AT TOP OF MEMORY
U 1B36, 2100,15     ;21039          DEC WR[0]                        ;DECREMENT BY 1 TO GET TOP VALID ADDR
U 1B37, BE14,15     ;21040          MOV WR[0] TO LS[T10]              ;SAVE IN LS
U 1B38, BE12,15     ;21041          MOV WR[0] TO LS[T9]               ;CURRENT ADDRESS IN T9
U 1B39, AE46,15     ;21042          SUB WR[3] FROM WR[0]              ;ALSO CALCULATE ADDRESS TO STOP AT
U 1B3A, 3E16,15     ;21043          MOV WR[0] TO LS[T11]              ;SAVE THIS IN LS
                    ;21044  LOOP.T33.6.8:
U 1B3B, 361A,95     ;21045          MOV LS[T13] TO WR[1]             ;EXPECTED DATA
U 1B3C, 1813,95     ;21046          MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
                    ;21047                                          ;SET UP TO READ ARRAY
U 1B3D, 3022,15     ;21048          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 1B3E, 0869,3C     ;21049          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 1B3F, 09B3,B4     ;21050          JMP [LOOP.T33.6.8]               ;ERROR LOOP IF ENABLED
                    ;21051
U 1B40, 9912,15     ;21052          MEM.REQ[WRITE.P] ADRS[T9] DT[BYTE]
                    ;21053                                          ;SET UP TO WRITE COMPLEMENT IN ARRAY
```

```
U 1B41, B218,15 ;21054          WRITE.MEM LS[T12]                ;WRITE COMP DATA IN THIS LS LOCATION
                ;21055
                ;21056
                ;21057          DEC LS[T14],                     ;DECREMENT COUNTER
U 1B42, FC1C,35 ;21058           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 1B43, 09B4,51 ;21059          JMP.IF[NEQ] TO [CONT.T33.6.8]    ;JMP IF COUNTER NOT YET ZERO
                ;21060
U 1B44, 89BB,AC ;21061          JSR [SETUP.T33.COUNTER&CALL.MONITOR] ;SET UP COUNTER AND CALL MONITOR
                ;21062                                           ; TO LET HIM KNOW WE'RE STILL ALIVE
                ;21063  CONT.T33.6.8:
                ;21064          SUB WR[3] FROM LS[T9],           ;TOGGLE BIT IN ADDRESS
U 1B45, 6B13,B5 ;21065           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 1B46, 89B3,B0 ;21066          JMP.IF[N.CLR] TO [LOOP.T33.6.8] ;IF NOT LESS THAN ADDRESS 0, REPEAT AT
                ;21067                                           ; NEXT ADDRESS
U 1B47, 3614,15 ;21068          MOV LS[T10] TO WR[0]             ;ELSE GET LAST ADDRESS WE STARTED AT
U 1B48, 2E44,15 ;21069          SUB WR[2] FROM WR[0]             ;DECREMENT ADDRESS
U 1B49, BE14,15 ;21070          MOV WR[0] TO LS[T10]             ;SAVE ADDRESS
U 1B4A, BE12,15 ;21071          MOV WR[0] TO LS[T9]              ;ADDRESS IN T9
                ;21072
                ;21073          CMP LS[T11] WITH WR[0],          ;IF ADDRESS = T11, WE'RE DONE
U 1B4B, 4E16,35 ;21074           DT(LONG)&SET.ALU.CC             ; SET CONDITION CODES
U 1B4C, 09B3,B1 ;21075          JMP.IF[NEQ] TO [LOOP.T33.6.8]    ;REPEAT TEST AT NEW ADDRESS IF NOT
                ;21076                                           ; DONE
                ;21077
U 1B4D, FC82,15 ;21078          DEC LS[ERROR.NUMBER]             ;DECREMENT ERROR NUMBER
U 1B4E, A3C1,95 ;21079          ROL WR[3]                        ;NEXT BIT TO TOGGLE AT MAX RATE
                ;21080          BIT LS[#4] WITH WR[3],           ;SEE IF BIT 2 HAS SET
U 1B4F, D945,B5 ;21081           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 1B50, 89B1,F9 ;21082          JMP.IF[BITS.CLR] TO [REPEAT.T33.3.7] ;REPEAT TEST WITH NEW TOGGLED BIT
                ;21083                                           ; IF BIT 2 HAS NOT SET
U 1B51, 5B00,14 ;21084          RETURN                           ;ELSE DONE
                ;21085
                ;21086  T33.9:
U 1B52, 89BB,CC ;21087          JSR [CALL.MONITOR]               ;CALL MONITOR TO LET HIM KNOW WE'RE
                ;21088                                           ; STILL ALIVE
U 1B53, B645,15 ;21089          MOV LS[#4] TO WR[2]              ;SET UP TO INCREMENT BY 4
U 1B54, 3625,95 ;21090          MOV LS[MM.LASTADDR+1] TO WR[3]   ;LAST ADDRESS + 1 IN WR FOR CMP
U 1B55, 41C1,95 ;21091          SUB LS[#3(H)] FROM WR[3]         ;ADDRESS TO STOP AT ON ASCENDING PASS
U 1B56, 6512,15 ;21092          CLR LS[T9]                       ;0 IN LS
U 1B57, FF12,15 ;21093          INC LS[T9]                       ;START AT ADDRESS 1 IN ARRAY (UNALIGNED)
                ;21094
                ;21095  LOOP.T33.9:
U 1B58, B618,95 ;21096          MOV LS[T12] TO WR[1]             ;EXPECTED DATA
U 1B59, 1813,F5 ;21097          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;21098                                           ;SET UP TO READ ARRAY
U 1B5A, 3022,15 ;21099          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 1B5B, 0869,3C ;21100          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 1B5C, 09B5,84 ;21101          JMP [LOOP.T33.9]                 ;ERROR LOOP IF ENABLED
                ;21102
U 1B5D, 9912,75 ;21103          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;21104                                           ;SET UP TO WRITE COMPLEMENT IN ARRAY
U 1B5E, 321A,15 ;21105          WRITE.MEM LS[T13]                ;WRITE COMP DATA IN THIS LOCATION
                ;21106
U 1B5F, EC13,15 ;21107          ADD WR[2] TO LS[T9]              ;INCREMENT BY 4
                ;21108          CMP WR[3] WITH LS[T9],           ;SEE IF DONE WITH ASCENDING PASS
```

```
U 1B60, 4F13,B5 ;21109          DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 1B61, 09B5,81 ;21110          JMP.IF[NEQ] TO [LOOP.T33.9]     ;IF NOT AT TOP OF MEMORY, REPEAT AT
                ;21111                                          ; NEXT ADDRESS
                ;21112                                          ; ELSE GOING TO START DESCENDING ADDRS
U 1B62, 89BB,CC ;21113          JSR [CALL.MONITOR]              ;CALL MONITOR TO LET HIM KNOW WE'RE
                ;21114                                          ; STILL ALIVE
                ;21115
U 1B63, FF82,15 ;21116          INC LS[ERROR.NUMBER]            ;ERROR A
U 1B64, 3624,15 ;21117          MOV LS[MM.LASTADDR+1] TO WR[0]  ;START AT TOP OF MEMORY
U 1B65, 41C0,15 ;21118          SUB LS[#3(H)] FROM WR[0]        ;DECREMENT BY 3 TO GET TOP VALID ADDR
U 1B66, BE12,15 ;21119          MOV WR[0] TO LS[T9]             ;CURRENT ADDRESS IN T9
U 1B67, B641,95 ;21120          MOV LS[#1] TO WR[3]             ;ADDRESS TO STOP AT
                ;21121  REPEAT.T33.A:
U 1B68, 6B13,15 ;21122          SUB WR[2] FROM LS[T9]           ;DECREMENT BY 4
                ;21123  LOOP.T33.A:
U 1B69, 361A,95 ;21124          MOV LS[T13] TO WR[1]            ;EXPECTED DATA
U 1B6A, 1813,F5 ;21125          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                ;21126                                          ;SET UP TO READ ARRAY
U 1B6B, 3022,15 ;21127          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1B6C, 0869,3C ;21128          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 1B6D, 89B6,94 ;21129          JMP [LOOP.T33.A]                ;ERROR LOOP IF ENABLED
                ;21130
U 1B6E, 9912,75 ;21131          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;21132                                          ;SET UP TO WRITE COMPLEMENT IN ARRAY
U 1B6F, B218,15 ;21133          WRITE.MEM LS[T12]               ;WRITE COMP DATA IN THIS LS LOCATION
                ;21134
                ;21135          CMP LS[T9] WITH WR[3],          ;ADDRESS TO STOP AT
U 1B70, CE13,B5 ;21136           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1B71, 09B6,81 ;21137          JMP.IF[NEQ] TO [REPEAT.T33.A]   ;IF NOT AT LOW ADDRESS, REPEAT AT
                ;21138                                          ; NEXT ADDRESS
                ;21139
U 1B72, 89BB,CC ;21140          JSR [CALL.MONITOR]              ;CALL MONITOR TO LET HIM KNOW WE'RE
                ;21141                                          ; STILL ALIVE
U 1B73, 5B00,14 ;21142          RETURN                          ;DONE
                ;21143
                ;21144
                ;21145  T33.B:
U 1B74, 3625,95 ;21146          MOV LS[MM.LASTADDR+1] TO WR[3]  ;LAST ADDRESS + 1 IN WR
U 1B75, C147,95 ;21147          SUB LS[#8] FROM WR[3]           ;SUBTRACT 8 DECIMAL
U 1B76, 4145,95 ;21148          SUB LS[#4] FROM WR[3]           ;SUBTRACT 4 DECIMAL FOR ADDRESS TO
                ;21149                                          ; STOP AT ON ASCENDING PASS
U 1B77, B649,15 ;21150          MOV LS[#10] TO WR[2]            ;INCREMENT BY 16 DECIMAL
U 1B78, 3644,15 ;21151          MOV LS[#4] TO WR[0]             ;4 IN WR
U 1B79, BE12,15 ;21152          MOV WR[0] TO LS[T9]             ;START AT ADDRESS 4 (TO MAXIMIZE CROSSING
                ;21153                                          ; PAGE BOUNDARIES)
                ;21154  LOOP.T33.B:
U 1B7A, B618,95 ;21155          MOV LS[T12] TO WR[1]            ;EXPECTED DATA
U 1B7B, 1E12,75 ;21156          MEM.REQ[OCTA.READ.P] ADRS[T9] DT[LONG]
                ;21157                                          ;SET UP TO READ ARRAY
U 1B7C, 3022,15 ;21158          MOV MEM.DATA TO WR[0]           ;GET RESULT  LONGWORD 1
U 1B7D, 38AE,15 ;21159          MOV MEM.DATA TO LS[T57]         ;GET RESULT  LONGWORD 2
U 1B7E, 38B0,15 ;21160          MOV MEM.DATA TO LS[T58]         ;GET RESULT  LONGWORD 3
U 1B7F, B8B2,15 ;21161          MOV MEM.DATA TO LS[T59]         ;GET RESULT  LONGWORD 4
                ;21162
U 1B80, 0869,3C ;21163          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
```

```
U 1B81, 09B7,A4 ;21164          JMP [LOOP.T33.B]                ;ERROR LOOP IF ENABLED
                ;21165
U 1B82, B618,95 ;21166          MOV LS[T12] TO WR[1]            ;EXPECTED DATA
U 1B83, B6AE,15 ;21167          MOV LS[T57] TO WR[0]            ;RECEIVED DATA  LONGWORD 2
U 1B84, 0869,3C ;21168          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 1B85, 09B7,A4 ;21169          JMP [LOOP.T33.B]                ;ERROR LOOP IF ENABLED
                ;21170
U 1B86, B618,95 ;21171          MOV LS[T12] TO WR[1]            ;EXPECTED DATA
U 1B87, B6B0,15 ;21172          MOV LS[T58] TO WR[0]            ;RECEIVED DATA  LONGWORD 3
U 1B88, 0869,3C ;21173          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 1B89, 09B7,A4 ;21174          JMP [LOOP.T33.B]                ;ERROR LOOP IF ENABLED
                ;21175
U 1B8A, B618,95 ;21176          MOV LS[T12] TO WR[1]            ;EXPECTED DATA
U 1B8B, 36B2,15 ;21177          MOV LS[T59] TO WR[0]            ;RECEIVED DATA  LONGWORD 4
U 1B8C, 0869,3C ;21178          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 1B8D, 09B7,A4 ;21179          JMP [LOOP.T33.B]                ;ERROR LOOP IF ENABLED
                ;21180
U 1B8E, 9C12,75 ;21181          MEM.REQ[OCTA.WRITE.P] ADRS[T9] DT[LONG]
                ;21182                                          ;SET UP TO WRITE COMPLEMENT DATA
U 1B8F, 321A,15 ;21183          WRITE.MEM LS[T13]               ;WRITE LONGWORD 1
U 1B90, 321A,15 ;21184          WRITE.MEM LS[T13]               ;WRITE LONGWORD 2
U 1B91, 321A,15 ;21185          WRITE.MEM LS[T13]               ;WRITE LONGWORD 3
U 1B92, 321A,15 ;21186          WRITE.MEM LS[T13]               ;WRITE LONGWORD 4
                ;21187
U 1B93, EC13,15 ;21188          ADD WR[2] TO LS[T9]             ;INCREMENT BY 16 DECIMAL
                ;21189          CMP WR[3] WITH LS[T9],          ;SEE IF DONE WITH ASCENDING PASS
U 1B94, 4F13,B5 ;21190           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1B95, 09B7,A1 ;21191          JMP.IF[NEQ] TO [LOOP.T33.B]     ;IF NOT AT TOP OF MEMORY, REPEAT AT
                ;21192                                          ; NEXT ADDRESS
                ;21193                                          ; ELSE GOING TO START DESCENDING ADDRS
U 1B96, 89BB,CC ;21194          JSR [CALL.MONITOR]              ;CALL MONITOR TO LET HIM KNOW WE'RE
                ;21195                                          ; STILL ALIVE
                ;21196
U 1B97, FF82,15 ;21197          INC LS[ERROR.NUMBER]            ;ERROR C
U 1B98, 3624,15 ;21198          MOV LS[MM.LASTADDR+1] TO WR[0]  ;START AT TOP OF MEMORY
U 1B99, C146,15 ;21199          SUB LS[#8] FROM WR[0]           ;DECREMENT BY 8
U 1B9A, 4144,15 ;21200          SUB LS[#4] FROM WR[0]           ;DECREMENT BY 4 TO GET TOP ADDRESS TO
                ;21201                                          ; ACCESS
U 1B9B, BE12,15 ;21202          MOV WR[0] TO LS[T9]             ;LOAD ADDRESS INTO T9
U 1B9C, 3645,95 ;21203          MOV LS[#4] TO WR[3]             ;ADDRESS TO STOP AT ON DESCENDING PASS
                ;21204
                ;21205  REPEAT.T33.C:
U 1B9D, 6B13,15 ;21206          SUB WR[2] FROM LS[T9]           ;DECREMENT BY 16 DECIMAL
                ;21207  LOOP.T33.C:
U 1B9E, 361A,95 ;21208          MOV LS[T13] TO WR[1]            ;EXPECTED DATA
U 1B9F, 1E12,75 ;21209          MEM.REQ[OCTA.READ.P] ADRS[T9] DT[LONG]
                ;21210                                          ;SET UP TO READ ARRAY
U 1BA0, 3022,15 ;21211          MOV MEM.DATA TO WR[0]           ;GET RESULT  LONGWORD 1
U 1BA1, 38AE,15 ;21212          MOV MEM.DATA TO LS[T57]         ;GET RESULT  LONGWORD 2
U 1BA2, 38B0,15 ;21213          MOV MEM.DATA TO LS[T58]         ;GET RESULT  LONGWORD 3
U 1BA3, B8B2,15 ;21214          MOV MEM.DATA TO LS[T59]         ;GET RESULT  LONGWORD 4
                ;21215
U 1BA4, 0869,3C ;21216          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 1BA5, 09B9,E4 ;21217          JMP [LOOP.T33.C]                ;ERROR LOOP IF ENABLED
                ;21218
```

```
U 1BA6, 361A,95 ;21219          MOV LS[T13] TO WR[1]             ;EXPECTED DATA
U 1BA7, B6AE,15 ;21220          MOV LS[T57] TO WR[0]             ;RECEIVED DATA  LONGWORD 2
U 1BA8, 0869,3C ;21221          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 1BA9, 09B9,E4 ;21222          JMP [LOOP.T33.C]                 ;ERROR LOOP IF ENABLED
                ;21223
U 1BAA, 361A,95 ;21224          MOV LS[T13] TO WR[1]             ;EXPECTED DATA
U 1BAB, B6B0,15 ;21225          MOV LS[T58] TO WR[0]             ;RECEIVED DATA  LONGWORD 3
U 1BAC, 0869,3C ;21226          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 1BAD, 09B9,E4 ;21227          JMP [LOOP.T33.C]                 ;ERROR LOOP IF ENABLED
                ;21228
U 1BAE, 361A,95 ;21229          MOV LS[T13] TO WR[1]             ;EXPECTED DATA
U 1BAF, 36B2,15 ;21230          MOV LS[T59] TO WR[0]             ;RECEIVED DATA  LONGWORD 4
U 1BB0, 0869,3C ;21231          JSR [CHECK.RESULT]               ;CHECK FOR ERRORS
U 1BB1, 09B9,E4 ;21232          JMP [LOOP.T33.C]                 ;ERROR LOOP IF ENABLED
                ;21233
U 1BB2, 9C12,75 ;21234          MEM.REQ[OCTA.WRITE.P] ADRS[T9] DT[LONG]
                ;21235                                           ;SET UP TO WRITE COMPLEMENT DATA
U 1BB3, B218,15 ;21236          WRITE.MEM LS[T12]                ;WRITE LONGWORD 1
U 1BB4, B218,15 ;21237          WRITE.MEM LS[T12]                ;WRITE LONGWORD 2
U 1BB5, B218,15 ;21238          WRITE.MEM LS[T12]                ;WRITE LONGWORD 3
U 1BB6, B218,15 ;21239          WRITE.MEM LS[T12]                ;WRITE LONGWORD 4
                ;21240
                ;21241          CMP LS[T9] WITH WR[3],           ;ADDRESS TO STOP AT
U 1BB7, CE13,B5 ;21242           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 1BB8, 09B9,D1 ;21243          JMP.IF[NEQ] TO [REPEAT.T33.C]    ;IF NOT AT LOW ADDRESS, REPEAT AT
                ;21244                                           ; NEXT ADDRESS
                ;21245
U 1BB9, 5B00,14 ;21246          RETURN                           ;DONE
                ;21247
                ;21248
                ;21249  ;
                ;21250  ; THIS ROUTINE CALLS THE MONITOR TO PREVENT A TIMEOUT.  IF ENTERED AT TOP, ALSO
                ;21251  ; SETS UP A COUNTER FOR BYTE ACCESS TEST.  THE COUNTER ENABLES US TO CALL THE
                ;21252  ; MONITOR AFTER EACH 1 MEG OF ACCESSES.
                ;21253  ;
                ;21254
                ;21255  SETUP.T33.COUNTER&CALL.MONITOR:
U 1BBA, B668,15 ;21256          MOV LS[#100000] TO WR[0]         ;SET UP COUNTER FOR 1 MILLION ACCESSES
                ;21257                                           ; WILL CALL MONITOR AFTER EACH 1 MEG
U 1BBB, 3E1C,15 ;21258          MOV WR[0] TO LS[T14]             ;STORE IN LS
                ;21259  CALL.MONITOR:
U 1BBC, E580,15 ;21260          CLR LS[CONTROL.STATUS]           ;CLEAR CONTROL AND STATUS WORD
U 1BBD, 10E0,15 ;21261          MISC [SET.CP.ATTN]               ;ISSUE CPU ATTN TO CONSOLE TO TELL
                ;21262                                           ; MONITOR THAT WE ARE STILL ALIVE
                ;21263  WAIT.T33.CALL:
U 1BBE, 89BB,E4 ;21264          JMP [WAIT.T33.CALL]              ;LOOP HERE WAITING FOR CONSOLE TO
                ;21265                                           ; RESPOND
U 1BBF, 5B00,14 ;21266          RETURN                           ;DONE
                ;21267
                ;21268  ;
                ;21269  ; THIS ROUTINE WRITES A BACKGROUND OF THE DATA IN T12 THROUGHOUT ALL AVAILABLE
                ;21270  ; MEMORY.
                ;21271  ;
                ;21272  BACK.T33:
U 1BC0, 3624,15 ;21273          MOV LS[MM.LASTADDR+1] TO WR[0]   ;LAST ADDRESS+1 IN WR FOR CMP
```

```
U 1BC1, B645,15 ;21274          MOV LS[#4] TO WR[2]              ;INCREMENT VALUE IN WR 2
U 1BC2, 6512,15 ;21275          CLR LS[T9]                       ;START AT ADDRESS 0                    U 1
                ;21276  BACK.T33.REP:
U 1BC3, 9912,75 ;21277          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                ;21278                                           ;SET UP TO WRITE BACKGROUND IN ARRAY   U 1
U 1BC4, B218,15 ;21279          WRITE.MEM LS[T12]                ;WRITE DATA IN ARRAY
U 1BC5, EC13,15 ;21280          ADD WR[2] TO LS[T9]              ;INCREMENT ADDRESS BY 4                U 1
                ;21281          CMP WR[0] WITH LS[T9],           ;SEE IF WRITTEN ALL MEMORY
U 1BC6, 4F12,35 ;21282           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 1BC7, 89BC,31 ;21283          JMP.IF[NEQ] TO [BACK.T33.REP]    ;REPEAT UNTIL ALL MEMORY WRITTEN       U 1
U 1BC8, 5B00,14 ;21284          RETURN                           ;DONE WITH BACKGROUND                  U 1
                ;21285                                                                                  U 1
                ;21286  ;                                                                               U 1
                ;21287  ; THIS ROUTINE INCREMENTS THE ERROR NUMBER BY 2                                  U 1
                ;21288  ;                                                                               U 1
                ;21289                                                                                  U 1
                ;21290  T33.INC.ERR:                                                                    U 1
U 1BC9, FF82,15 ;21291          INC LS[ERROR.NUMBER]             ;INCREMENT ERROR NUMBER                U 1
U 1BCA, FF82,15 ;21292          INC LS[ERROR.NUMBER]             ;INCREMENT AGAIN
U 1BCB, 5B00,14 ;21293          RETURN                           ;DONE                                  U 1
                ;21294                                                                                  U 1
                ;21295                                                                                  U 1
                ;21296  END.T33:                                                                        U 1
                                                                                                        U 1
                                                                                                        U 1
                                
```

```
;21297  .page   "UBE MICRO DIAGNOSTICS"
;21298
;21299  ;         The UBE micro diagnostics are designed to check out the MCT UNIBUS
;21300  ;       hardware that could not be tested in previous tests without a UNIBUS
;21301  ;       device.  The micro diagnostics are NOT designed as an exerciser of
;21302  ;       UNIBUS logic.
;21303  ;         The first test sizes for a UBE by writing to the UNIBUS address
;21304  ;       that would respond if a UBE were installed and configured as described
;21305  ;       in the next section below.  The diagnostic then checks the NXM bit in
;21306  ;       the MCT CSR 1.  If NXM is set, meaning no response was received from
;21307  ;       the write to UNIBUS device, then the remaining UBE tests are skipped.
;21308  ;       If NXM is not set, meaning a response from the write was received, then
;21309  ;       the UBE tests begin.  A message is printed in either case to inform the
;21310  ;       operator as to the result of the sizing test.
;21311  ;         If the UBE is not found, and one is installed and configured correct-
;21312  ;       ly, the operator can force the sizing test to loop so that the problem
;21313  ;       can be debugged.  This is accomplished by setting the SER flag (SE SE)
;21314  ;       and executing the test.  This procedure is described in detail in the
;21315  ;       sizing test description.
;21316  ;         The last UBE test is not really a test at all, but a debug tool.  It
;21317  ;       is not run unless specifically executed via a DI TE 43 command.  The
;21318  ;       test simply reports the state of the UBE registers.  See the test for a
;21319  ;       more detailed description.
;21320
;21321
```

```
;21322  .page   "UBE SETUP AND REGISTER DESCRIPTION"
;21323
;21324  ;
;21325  ;         These tests are designed to run with at least 1 UBE installed in the
;21326  ;       backplane responding to addresses FFFFF000(H) through FFFFF00E(H)
;21327  ;       (770000 through 770016 in octal).  More than 1 UBE may be installed,
;21328  ;       but this test will only use the one responding to the above addresses.
;21329  ;       The interrupt vector on this UBE must be set to respond with address
;21330  ;       510(octal).
;21331  ;         The switch settings for these requirements are as follows:
;21332  ;
;21333  ;       LOCATION  S1      S2      S3      S4      S5      S6      S7      S8
;21334  ;
;21335  ;         E125    ON      ON      ON      ON      ON      ON      ON      ON
;21336  ;         E88     ON      ON      X       ON      OFF     OFF     ON      OFF
;21337  ;
;21338  ;         X=Don't Care.  E125 is the address switch, E88 is the vector switch.
;21339  ;
;21340  ;
;21341  ;         The UBE registers are loaded with a WRITE.P to the corresponding
;21342  ;       addresses.  The addresses is interpreted as a UNIBUS physical address
;21343  ;       by the memory controller.  Following is a brief description of each
;21344  ;       register and its corresponding hex address.  See the UBE spec for more
;21345  ;       detailed information.
;21346  ;
;21347  ;         DATA REG (BEDB) - Hex Address FFFFF000
;21348  ;
;21349  ;               This register contains the 16 bit data to be written to the
;21350  ;               array during a DATO operation by the UBE, or the data read
;21351  ;               from the array on a DATI operation by the UBE.
;21352  ;
;21353  ;         CYCLE COUNT REG (BECC) - Hex Address FFFFF002
;21354  ;
;21355  ;               This register is loaded with the 2's complement of the number
;21356  ;               of transfers that the UBE is to perform.  For most of these
;21357  ;               tests, this register is loaded with -1, to do one transfer.
;21358  ;
;21359  ;         ADDR REG (BEBA) - Hex Address FFFFF004
;21360  ;
;21361  ;               This register is loaded with the array address to be accessed
;21362  ;               on either a write or a read from the UBE.  It is incremented
;21363  ;               by 2 after each transfer (incremented by one on a DATOB cycle).
;21364  ;
;21365  ;         CONTROL REG 1 (BECR1) - Hex Address FFFFF006
;21366  ;
;21367  ;               This register contains the control information for the UBE.  It
;21368  ;               tells the UBE what function it is to do.  Bit 15 is an error
;21369  ;               bit which sets if any error bits in control reg 2 are set.  Bit
;21370  ;               0 is the GO bit, which is set to a 1, by writing the register,
;21371  ;               to start the UBE.  Below is a synopsis of the bit functions.
;21372  ;               See the UBE spec for a more detailed description.
```

```
;21373  .page
;21374  ;
;21375  ;          15     14     13    12   11  10   9   8   7   6    5  4  3  2  1  0
;21376  ;
;21377  ;        ┌──────┬──────┬──────┬──────┬──────┬──────┬──────┬──────┬──────┬──────┬──────┬──────┬──────┬──────┬──────┬──────┐
;21378  ;        │ERR   │DATO  │INH   │DATA  │FUN   │FUN   │      │      │      │INT   │      │BR    │BR    │BR    │BR    │      │
;21379  ;        │FLAG  │ OR   │DATIP │ OR   │  B   │  A   │C01   │C00   │RDY   │ ON   │NPR   │ 7    │ 6    │ 5    │ 4    │GO    │
;21380  ;        │      │DATOB │ ROL  │CYCLE │      │      │      │      │      │DONE  │      │      │      │      │      │      │
;21381  ;        └──────┴──────┴──────┴──────┴──────┴──────┴──────┴──────┴──────┴──────┴──────┴──────┴──────┴──────┴──────┴──────┘
;21382  ;
;21383  ;                C01-C00:   0=DATI 1=DATIP 2=DATO 3=DATOB
;21384  ;                FUNB-FUNA: 0=DO INTERRUPT 1=DO WORD XFER 2=DO LONGWORD XFER
;21385  ;                           3=DROP BUS
;21386  ;
;21387  ;          CLEAR ERR ADDRESS (CLEAR.ERR.ADDR) - Hex Address FFFFF00A
;21388  ;
;21389  ;                This is not an actual register.  When a DATO to this address
;21390  ;                is executed, any error bits set in control reg 2 are cleared.
;21391  ;
;21392  ;          CONTROL REG 2 (BECR2) - Hex Address FFFFF010
;21393  ;
;21394  ;                This register contains some control bits and error indicators.
;21395  ;                Bit 13 and bit 2 are the only control bits used by these tests.
;21396  ;                Bit 13 is the cycle count overflow bit and sets when the re-
;21397  ;                quested number of cycles is completed.  Bit 2 is the inhibit
;21398  ;                increment of address and cycle count reg bit and prevents the
;21399  ;                UBE from incrementing the cycle count and address registers
;21400  ;                after each transfer.  This bit is used in the test that check
;21401  ;                the lock function to cause the UBE to issue continuous NPR
;21402  ;                transfers until stopped by clearing this bit. The other control
;21403  ;                bits are always clear for these tests.  Below is a diagram of
;21404  ;                the error indicator bits (bits 11 through 5) which are given
;21405  ;                in the error report if a UBE error occurs:
;21406  ;
;21407  ;                  11    10    9    8     7     6    5
;21408  ;                ┌──────┬──────┬──────┬──────┬──────┬──────┬──────┐
;21409  ;                │NO    │      │Q     │NO    │NO,   │BUS   │Q     │
;21410  ;                │INTR  │OTOG  │A     │SSYN  │NO    │LAT   │G     │
;21411  ;                │SSYN  │      │L     │      │SACK  │      │B     │
;21412  ;                └──────┴──────┴──────┴──────┴──────┴──────┴──────┘
;21413  ;
;21414  ;                  11 = No SSYN received after INTR sent to CPU
;21415  ;                  10 = No grant or more than one grant (Other Than One Grant)
;21416  ;                   9 = Wrong Address Lines when addr sent compared to addr recv
;21417  ;                   8 = No SSYN received after MSYN
;21418  ;                   7 = No CPU timeout when sack was not sent after getting grant
;21419  ;                   6 = Bus latency error
;21420  ;                   7 = Wrong Grant Back
```

```
;21421  .page   '' UBE Tests''
;21422  .toc    ''  TEST 34 UBE Present Test''
;21423  ;++
;21424  ;
;21425  ; Test Description:
;21426  ;
;21427  ;       This test checks to see if a UBE is present in the NEBULA system.  If
;21428  ;       a UBE is found, a message is printed (UBE PRESENT) and execution passes
;21429  ;       on to the next test.  If a UBE is not found, a message is printed (UBE
;21430  ;       NOT PRESENT) and the UBE test is skipped.
;21431  ;         The test sizes for a UBE by doing a READ.P from address FFF000 (the
;21432  ;       UBE data register) and checking for NXM set in CSR 1 of the memory
;21433  ;       controller.  NXM set indicates a timeout occurred, signifying that a
;21434  ;       UBE was not found.  NXM clear indicates that a UBE was found and re-
;21435  ;       sponded.
;21436  ;
;21437  ; Assumptions:
;21438  ;
;21439  ;       It is assumed that all other micro-diagnostics have run successfully.
;21440  ;       If a UBE is present, it is assumed to be good.
;21441  ;
;21442  ;       NOTE:  The SP (special bit) flag must be clear unless looping
;21443  ;       on this test is desired (see debug)!
;21444  ;
;21445  ; Test Steps:
;21446  ;
;21447  ;       1)  Issue DCLO to init UBE if present.
;21448  ;       2)  Issue MEM.REQ[READ.P] to the UBE data buffer.  Complete the memory
;21449  ;           cycle by issueing MOV MEM.DATA TO WR[0].
;21450  ;       3)  Check SER flag.  If set, repeat step 1 repeatedly (allows looping).
;21451  ;       4)  Read CSR 1 and check the NXM bit.  If clear, report UBE PRESENT.
;21452  ;           If set, report UBE NOT PRESENT.
;21453  ;       5)  If UBE not present, skip to end of section.  Else go on to next
;21454  ;           test.
;21455  ;
;21456  ; Errors:
;21457  ;
;21458  ;       Error 1 - Result of sizing for UBE differed from what APT said was
;21459  ;                 there (this error only possible under APT).
;21460  ;
;21461  ; Debug:
;21462  ;
;21463  ;       If a UBE is present but not found by this test, the READ to the UBE
;21464  ;       data reg can be looped by typing SE SP (set special control bit), SE
;21465  ;       LO (set loop on error to prevent a timeout),and starting this test over
;21466  ;       again.  Remember to CL SP when you want to continue on to other
;21467  ;       tests.  CNTL C or CNTL P must be typed to exit this loop.  It is poss-
;21468  ;       sible for the MCT to hang when exiting the loop.  Type I (INIT) to free
;21469  ;       the MCT if necessary.
;21470  ;         While looping, look for SSYN coming back to the receiver and through
;21471  ;       the 2 latches as L SSYN H.  If this is OK, check the signal going to
;21472  ;       the BEN 1 PAL.  If there is no SSYN, suspect a problem with the etch to
;21473  ;       the fingers  associated with the UNIBUS address lines, MSYN going out,
;21474  ;       or SSYN coming in.
;21475  ;
```

```
                ;21476  ;--
                ;21477
                ;21478  T.34:
U 1BCC, B65E,15 ;21479          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;21480                                          ; STATUS WORD
U 1BCD, 3E80,15 ;21481          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;21482                                          ; BIT 15 INDICATES START OF TEST TO
                ;21483                                          ; CONSOLE
U 1BCE, 10E0,15 ;21484          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;21485  WAIT.T34.0:
U 1BCF, 89BC,F4 ;21486          JMP [WAIT.T34.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;21487                                          ; RESPOND
                ;21488
U 1BD0, 0A1B,8C ;21489          JSR [ISSUE.DCLO]                ;ISSUE DCLO TO INIT UBE
                ;21490  LOOP.T34.1:
U 1BD1, 98B5,B5 ;21491          MEM.REQ[READ.P] ADRS[BEDB] DT[WORD]
                ;21492                                          ;SET UP TO READ DATA BUFFER OF UBE
U 1BD2, 3022,15 ;21493          MOV MEM.DATA TO WR[0]           ;COMPLETE MEMORY REFERENCE
U 1BD3, 3690,15 ;21494          MOV LS[ERROR.CONTROL] TO WR[0]  ;GET ERROR CONTROL WORD
                ;21495          BIT LS[SPECIAL] WITH WR[0],     ;SEE IF SE SP (SET SPECIAL BIT) IS
                ;21496                                          ; ACTIVE
U 1BD4, D94E,35 ;21497           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1BD5, 89BD,11 ;21498          JMP.IF[BIT.SET] TO [LOOP.T34.1] ;LOOP IF SE SP IS ACTIVE
                ;21499
U 1BD6, 9D45,75 ;21500          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;21501                                          ;SET UP TO READ CSR 1
U 1BD7, 3022,15 ;21502          MOV MEM.DATA TO WR[0]           ;GET CSR 1 CONTENTS
U 1BD8, E50E,15 ;21503          CLR LS[UBE.FOUND]               ;CLEAR LS 7
                ;21504          BIT LS[NXM] WITH WR[0],         ;SEE IF NXM IS SET
U 1BD9, D960,35 ;21505           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1BDA, 2F80,15 ;21506          CLR WR[0]                       ;SETUP FOR NO UBE
U 1BDB, 09BD,F1 ;21507          JMP.IF[BIT.SET] TO [CHECK.APT]  ;DON'T SET BIT IN LS 7 IF NXM WAS SET
U 1BDC, 7F0E,15 ;21508          INC LS[UBE.FOUND]               ;ELSE SET BIT 0 IN LS 7
U 1BDD, 3650,15 ;21509          MOV LS[BIT8] TO WR[0]           ;SET BIT 8 IN WR
U 1BDE, 6D0E,15 ;21510          BIS WR[0] TO LS[UBE.FOUND]      ;SET BIT 8 IN LS IF UBE FOUND (NO NXM)
                ;21511  CHECK.APT:
U 1BDF, 3640,95 ;21512          MOV LS[#1] TO WR[1]             ;PUT A 1 IN WR
U 1BE0, 3E82,95 ;21513          MOV WR[1] TO LS[ERROR.NUMBER]   ;ERROR 1 (APT ONLY)
U 1BE1, B690,95 ;21514          MOV LS[ERROR.CONTROL] TO WR[1]  ;GET ERROR CONTROL WORD
                ;21515          BIT LS[APT.PRESENT] WITH WR[1], ;SEE IF UNDER APT
U 1BE2, D94A,B5 ;21516           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1BE3, 89BE,A9 ;21517          JMP.IF[BITS.CLR] TO [T34.PRINT] ;IF NOT APT, GO PRINT SIZING RESULT
                ;21518
U 1BE4, 3692,95 ;21519          MOV LS[APT.PARAM] TO WR[1]      ;ELSE GET APT PARAMETERS
U 1BE5, DF29,15 ;21520          MCOM LS[#FFFF] TO WR[2]         ;PUT FFFF0000 IN WR 2
U 1BE6, 2F44,95 ;21521          BIC WR[2] TO WR[1]              ;CLEAR UPPER WORD OF APT PARAM
U 1BE7, C526,95 ;21522          BIC LS[#FF] TO WR[1]            ;CLEAR LOW BYTE OF APT PARAM
U 1BE8, 0869,3C ;21523          JSR [CHECK.RESULT]              ;REPORT ERROR IF APT HARDWARE CONFIG
                ;21524                                          ; DIFFERENT FROM SIZING RESULT
U 1BE9, 89BE,E4 ;21525          JMP [END.T34]                   ;NO ERROR LOOP UNDER APT. TEST DONE
                ;21526
                ;21527  T34.PRINT:
U 1BEA, 364C,95 ;21528          MOV LS[PRINT.UBE] TO WR[1]      ;SET BIT 6 IN WR
U 1BEB, BE80,95 ;21529          MOV WR[1] TO LS[CONTROL.STATUS] ;SET BIT 6 IN CONTROL AND STATUS WORD
U 1BEC, 10E0,15 ;21530          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
```

```
                    ;21531  WAIT.T34.1:                                                                    U 1
U 1BED, 89BE,D4     ;21532          JMP [WAIT.T34.1]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;21533                                          ; RESPOND
                    ;21534  END.T34:
                    ;21535          MOV LS[UBE.FOUND] TO WR[0],     ;GET LS 7 RESULT                    U 1
U 1BEE, 360E,35     ;21536           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES           U 1
U 1BEF, 8A17,99     ;21537          JMP.IF[BITS.CLR] TO [END.SECTION] ;SKIP REST OF TESTS IF UBE NOT FOUND
                    ;21538                                                                              U 1
                    ;21539                                                                              U 1
                                                                                                        U 1
                                                                                                        U 1
                                                                                                        U 1
                                                                                                        U 1
                                                                                                        U 1
```

```
;21540  .page   "  TEST 35 Write/Read UBE Registers and DC LO Test (M8391 MCT Module)"
;21541  ;++
;21542  ;
;21543  ; Test Description:
;21544  ;
;21545  ;       This test checks that each of four registers in the UBE can be written
;21546  ;       and read correctly.  Some bits in control register 1 are not directly
;21547  ;       writable and will not be tested in this test.  The data reg (BEDB) will
;21548  ;       be accessed first.  Data patterns will be all 0's, all 1's, and a
;21549  ;       shifting 1 pattern.  This will verify the integrity of the data lines
;21550  ;       and the MCT micro code associated with writes and reads to the UNIBUS.
;21551  ;         The cycle count and address registers will be checked next using all
;21552  ;       1's and all 0's data.  Control reg 1 will be checked with all 1's and
;21553  ;       all zeros except for bit 0 (the GO bit will be kept at 0),and some bits
;21554  ;       will be masked out.  Control reg 2 will not be checked as most of the
;21555  ;       bits are read only, and the previous tests are sufficient to check the
;21556  ;       MCT logic and micro-code.
;21557  ;         The last test checks that UBS DC LO is getting out to the UBE.  UBS
;21558  ;       DC LO clears certain bits in the UBE registers.  The UBE ADDR REG is
;21559  ;       one such register.  The UBE ADDR REG will be written with ones and a
;21560  ;       UBS DC LO will be issued (via a call to the monitor to assert this
;21561  ;       signal and then drop it).  After the DC LO sequence, the UBE ADDR REG
;21562  ;       will be checked to assure that it was cleared.
;21563  ;
;21564  ; Assumptions:
;21565  ;
;21566  ;       It is assumed that all other micro-diagnostics have run successfully.
;21567  ;       It is also assumed that the UBE present is known to be good.
;21568  ;
;21569  ; Test Steps:
;21570  ;
;21571  ;       1)  Set up error mask, error number, and module number in LS (for
;21572  ;           error printouts) and clear previous error number in LS.  Also
;21573  ;           issue DCLO to clear error bits in the UBE.
;21574  ;       2)  Issue WRITE.P to the UBE data reg (BEDB) with all 0's data.  Read
;21575  ;           the data reg and verify contents.
;21576  ;       3)  Repeat step 2 with all 1's data.
;21577  ;       4)  Repeat step 2 with shifting 1's data.
;21578  ;       5)  Repeat steps 2 and 3 on the Cycle Count reg (BECC) and the Addr
;21579  ;           reg (BEBA).
;21580  ;       6)  Repeat steps 2 and 3 on Control Reg 1, but do not set the GO bit
;21581  ;           (bit 0).  Also mask out bits 7 and 15 (RDY and ERR) as they are
;21582  ;           not directly readable.
;21583  ;       7)  Write all 1's in the UBE ADDR REG.  Issue command to the monitor
;21584  ;           to assert UBS DC LO and then deassert it.  Check the ADDR REG
;21585  ;           for all 0's.
;21586  ;
;21587  ; Errors:
;21588  ;
;21589  ;       Error 1 - Write/Read to UBE Data Buffer failed with data of all zeros.
;21590  ;       Error 2 - Write/Read to UBE Data Buffer failed with data of all ones.
;21591  ;       Error 3 - Write/Read to UBE Data Buffer failed with data of shifting
;21592  ;                 ones.
;21593  ;       Error 4 - Write/Read to UBE Cycle Count Reg failed with data of all
;21594  ;                 zeros.
```

```
                    ;21595  ;       Error 5 - Write/Read to UBE Cycle Count Reg failed with data of all
                    ;21596  ;                 ones.
                    ;21597  ;       Error 6 - Write/Read to UBE Addr Reg failed with data of all zeros.
                    ;21598  ;       Error 7 - Write/Read to UBE Addr Reg failed with data of all ones.
                    ;21599  ;       Error 8 - Write/Read to UBE Control Reg 1 failed with data of all zeros.
                    ;21600  ;       Error 9 - Write/Read to UBE Control Reg 1 failed with data of all ones.
                    ;21601  ;       Error A - UBS DC LO failed to clear the UBE ADDR REG.
                    ;21602  ;
                    ;21603  ; Debug:
                    ;21604  ;
                    ;21605  ;       Error 1 and 2 - This error most likely indicates a problem with the
                    ;21606  ;       data lines as the leave the fingers of the fingers of the WCS module
                    ;21607  ;       or in the CP WRITE or READ TO UNIBUS micro-code on the MCT module.
                    ;21608  ;       While looping on the error, check the data lines going out of the
                    ;21609  ;       WCS module.  Trace back any problems to the MCT module if necessary.
                    ;21610  ;
                    ;21611  ;       Error 3 - Same as error 1 and 2 except suspect a short between the
                    ;21612  ;       data lines as they leave the WCS module.
                    ;21613  ;
                    ;21614  ;       Errors 4 through 9 - These errors are unlikely since the UBE is assumed
                    ;21615  ;       to be good.  However the problem could be in the lower bits of the
                    ;21616  ;       address lines.  The address bits that change from the first 3 errors
                    ;21617  ;       are as follows:  The Cycle Count reg has address bit 1 asserted, the
                    ;21618  ;       Addr reg has bit 2 asserted, and Control Reg 1 has bits 3 and 1 assert-
                    ;21619  ;       ed.
                    ;21620  ;
                    ;21621  ;       Error A - This error indicates that DC LO is not getting to the UBE
                    ;21622  ;       over the backplane.  UBS DC LO originates on the WCS module.
                    ;21623  ;
                    ;21624  ;--
                    ;21625
                    ;21626  T.35:
U 1BF0, B65E,15     ;21627          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;21628                                          ; STATUS WORD
U 1BF1, 3E80,15     ;21629          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;21630                                          ; BIT 15 INDICATES START OF TEST TO
                    ;21631                                          ; CONSOLE
U 1BF2, 10E0,15     ;21632          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;21633  WAIT.T35.0:
U 1BF3, 89BF,34     ;21634          JMP [WAIT.T35.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;21635                                          ; RESPOND
                    ;21636
U 1BF4, 0A1B,8C     ;21637          JSR [ISSUE.DCLO]                ;ISSUE DCLO TO UBE TO CLEAR ERR BITS
U 1BF5, 65A0,15     ;21638          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 1BF6, 5F28,15     ;21639          MCOM LS[#FFFF] TO WR[0]         ;FFFF0000 IN WR
U 1BF7, 3E8A,15     ;21640          MOV WR[0] TO LS[ERROR.MASK]     ;SET UP TO CHECK BITS 0-15 ONLY
U 1BF8, 3644,15     ;21641          MOV LS[MCT] TO WR[0]            ;SET UP MODULE CODE. BIT 2 SET IN WR
U 1BF9, C740,15     ;21642          BIS LS[WCS] TO WR[0]            ;SET BIT 0 FOR WCS MODULE
U 1BFA, 3E8C,15     ;21643          MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                    ;21644                                          ; MCT OR WCS BOARD
U 1BFB, B640,15     ;21645          MOV LS[#1] TO WR[0]             ;1 IN WR
U 1BFC, BE82,15     ;21646          MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
                    ;21647
                    ;21648  LOOP.T35.1:
U 1BFD, 19B4,35     ;21649          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
```

```
                 ;21650                                          ;SET UP TO WRITE DATA BUFFER OF UBE
U 1BFE, B29C,15  ;21651          WRITE.MEM LS[ZERO]              ;WRITE ALL ZEROS IN DATA BUFFER
U 1BFF, 98B5,B5  ;21652          MEM.REQ[READ.P] ADRS[BEDB] DT[WORD]
                 ;21653                                          ;SET UP TO READ DATA BUFFER OF UBE
U 1C00, 3022,15  ;21654          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF DATA BUFFER
U 1C01, B69C,95  ;21655          MOV LS[ZERO] TO WR[1]           ;EXPECTED DATA
U 1C02, 0869,3C  ;21656          JSR [CHECK.RESULT]              ;CHECK FOR ALL ZEROS
U 1C03, 09BF,D4  ;21657          JMP [LOOP.T35.1]                ;ERROR LOOP IF ENABLED
                 ;21658
U 1C04, FF82,15  ;21659          INC LS[ERROR.NUMBER]            ;ERROR 2
                 ;21660  LOOP.T35.2:
U 1C05, 19B4,35  ;21661          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                 ;21662                                          ;SET UP TO WRITE DATA BUFFER OF UBE
U 1C06, 329E,15  ;21663          WRITE.MEM LS[ONES]              ;WRITE ALL ONES IN DATA BUFFER
U 1C07, 98B5,B5  ;21664          MEM.REQ[READ.P] ADRS[BEDB] DT[WORD]
                 ;21665                                          ;SET UP TO READ DATA BUFFER OF UBE
U 1C08, 3022,15  ;21666          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF DATA BUFFER
U 1C09, 369E,95  ;21667          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 1C0A, 0869,3C  ;21668          JSR [CHECK.RESULT]              ;CHECK FOR ALL ONES
U 1C0B, 09C0,54  ;21669          JMP [LOOP.T35.2]                ;ERROR LOOP IF ENABLED
                 ;21670
U 1C0C, FF82,15  ;21671          INC LS[ERROR.NUMBER]            ;ERROR 3
U 1C0D, E5F8,15  ;21672          CLR LS[OS]                      ;CLEAR INDEX
                 ;21673  LOOP.T35.3:
U 1C0E, 19B4,35  ;21674          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                 ;21675                                          ;SET UP TO WRITE DATA BUFFER OF UBE
U 1C0F, B2F6,15  ;21676          WRITE.MEM LS[SHIFT.OS(4-0)]     ;WRITE SHIFTING 1'S IN DATA BUFFER
U 1C10, 98B5,B5  ;21677          MEM.REQ[READ.P] ADRS[BEDB] DT[WORD]
                 ;21678                                          ;SET UP TO READ DATA BUFFER OF UBE
U 1C11, 3022,15  ;21679          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF DATA BUFFER
U 1C12, B6F6,95  ;21680          MOV LS[SHIFT.OS(4-0)] TO WR[1]  ;EXPECTED DATA
U 1C13, 0869,3C  ;21681          JSR [CHECK.RESULT]              ;CHECK FOR SHIFTING ONES PATTERN
U 1C14, 89C0,E4  ;21682          JMP [LOOP.T35.3]                ;ERROR LOOP IF ENABLED
                 ;21683
U 1C15, 7FF8,15  ;21684          INC LS[OS]                      ;INCREMENT INDEX FOR NEXT PATTERN
U 1C16, 36F9,15  ;21685          MOV LS[OS] TO WR[2]             ;GOING TO CHECK IF ALL PATTERNS ARE DONE
                 ;21686          BIT LS[BIT5] WITH WR[2],        ;SEE IF OS HAS INCREMENTED TO 20(H)
U 1C17, D94B,35  ;21687           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 1C18, 09C0,E9  ;21688          JMP.IF[BITS.CLR] TO [LOOP.T35.3] ;REPEAT WITH NEXT PATTERN IF NOT DONE
                 ;21689
U 1C19, FF82,15  ;21690          INC LS[ERROR.NUMBER]            ;ERROR 4
U 1C1A, 3644,15  ;21691          MOV LS[MCT] TO WR[0]            ;SET UP MODULE CODE. BIT 2 SET IN WR
U 1C1B, 3E8C,15  ;21692          MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                 ;21693                                          ; MCT BOARD
                 ;21694  LOOP.T35.4:
U 1C1C, 99B6,35  ;21695          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                 ;21696                                          ;SET UP TO WRITE CYCLE COUNT REG OF UBE
U 1C1D, B29C,15  ;21697          WRITE.MEM LS[ZERO]              ;WRITE ALL ZEROS IN CYCLE COUNT REG
U 1C1E, 18B7,B5  ;21698          MEM.REQ[READ.P] ADRS[BECC] DT[WORD]
                 ;21699                                          ;SET UP TO READ CYCLE COUNT REG OF UBE
U 1C1F, 3022,15  ;21700          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CYCLE COUNT REG
U 1C20, B69C,95  ;21701          MOV LS[ZERO] TO WR[1]           ;EXPECTED DATA
U 1C21, 0869,3C  ;21702          JSR [CHECK.RESULT]              ;CHECK FOR ALL ZEROS
U 1C22, 89C1,C4  ;21703          JMP [LOOP.T35.4]                ;ERROR LOOP IF ENABLED
                 ;21704
```

```
U 1C23, FF82,15 ;21705          INC LS[ERROR.NUMBER]            ;ERROR 5
                ;21706  LOOP.T35.5:
U 1C24, 99B6,35 ;21707          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                ;21708                                          ;SET UP TO WRITE CYCLE COUNT REG OF UBE
U 1C25, 329E,15 ;21709          WRITE.MEM LS[ONES]              ;WRITE ALL ONES IN CYCLE COUNT REG
U 1C26, 18B7,B5 ;21710          MEM.REQ[READ.P] ADRS[BECC] DT[WORD]
                ;21711                                          ;SET UP TO READ CYCLE COUNT REG OF UBE
U 1C27, 3022,15 ;21712          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CYCLE COUNT REG
U 1C28, 369E,95 ;21713          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 1C29, 0869,3C ;21714          JSR [CHECK.RESULT]              ;CHECK FOR ALL ONES
U 1C2A, 09C2,44 ;21715          JMP [LOOP.T35.5]                ;ERROR LOOP IF ENABLED
                ;21716
U 1C2B, FF82,15 ;21717          INC LS[ERROR.NUMBER]            ;ERROR 6
                ;21718  LOOP.T35.6:
U 1C2C, 19B8,35 ;21719          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                ;21720                                          ;SET UP TO WRITE ADDR REG OF UBE
U 1C2D, B29C,15 ;21721          WRITE.MEM LS[ZERO]              ;WRITE ALL ZEROS IN ADDR REG
U 1C2E, 98B9,B5 ;21722          MEM.REQ[READ.P] ADRS[BEBA] DT[WORD]
                ;21723                                          ;SET UP TO READ ADDR REG OF UBE
U 1C2F, 3022,15 ;21724          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF ADDR REG
U 1C30, B69C,95 ;21725          MOV LS[ZERO] TO WR[1]           ;EXPECTED DATA
U 1C31, 0869,3C ;21726          JSR [CHECK.RESULT]              ;CHECK FOR ALL ZEROS
U 1C32, 89C2,C4 ;21727          JMP [LOOP.T35.6]                ;ERROR LOOP IF ENABLED
                ;21728
U 1C33, FF82,15 ;21729          INC LS[ERROR.NUMBER]            ;ERROR 7
                ;21730  LOOP.T35.7:
U 1C34, 19B8,35 ;21731          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                ;21732                                          ;SET UP TO WRITE ADDR REG OF UBE
U 1C35, 329E,15 ;21733          WRITE.MEM LS[ONES]              ;WRITE ALL ONES IN ADDR REG
U 1C36, 98B9,B5 ;21734          MEM.REQ[READ.P] ADRS[BEBA] DT[WORD]
                ;21735                                          ;SET UP TO READ ADDR REG OF UBE
U 1C37, 3022,15 ;21736          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF ADDR REG
U 1C38, 369E,95 ;21737          MOV LS[ONES] TO WR[1]           ;EXPECTED DATA
U 1C39, 0869,3C ;21738          JSR [CHECK.RESULT]              ;CHECK FOR ALL ONES
U 1C3A, 89C3,44 ;21739          JMP [LOOP.T35.7]                ;ERROR LOOP IF ENABLED
                ;21740
U 1C3B, FF82,15 ;21741          INC LS[ERROR.NUMBER]            ;ERROR 8
U 1C3C, 364E,15 ;21742          MOV LS[RDY] TO WR[0]            ;SET BIT 7 IN WR
U 1C3D, C75E,15 ;21743          BIS LS[ERR] TO WR[0]            ;ALSO SET BIT 15 IN WR
U 1C3E, 6D8A,15 ;21744          BIS WR[0] TO LS[ERROR.MASK]     ;CHANGE ERROR MASK.  DON'T CHECK BIT
                ;21745                                          ; 7 OR 15 (ERR AND RDY BITS)
                ;21746  LOOP.T35.8:
U 1C3F, 99BA,35 ;21747          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                ;21748                                          ;SET UP TO WRITE CONTROL REG 1 OF UBE
U 1C40, B29C,15 ;21749          WRITE.MEM LS[ZERO]              ;WRITE ALL ZEROS IN CONTROL REG 1
U 1C41, 18BB,B5 ;21750          MEM.REQ[READ.P] ADRS[BECR1] DT[WORD]
                ;21751                                          ;SET UP TO READ CONTROL REG 1 OF UBE
U 1C42, 3022,15 ;21752          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CONTROL REG 1
U 1C43, B69C,95 ;21753          MOV LS[ZERO] TO WR[1]           ;EXPECTED DATA
U 1C44, 0869,3C ;21754          JSR [CHECK.RESULT]              ;CHECK FOR ALL ZEROS
U 1C45, 09C3,F4 ;21755          JMP [LOOP.T35.8]                ;ERROR LOOP IF ENABLED
                ;21756
U 1C46, FF82,15 ;21757          INC LS[ERROR.NUMBER]            ;ERROR 9
U 1C47, DF40,15 ;21758          MCOM LS[GO] TO WR[0]            ;ALL ONES EXCEPT BIT 0 IN WR
U 1C48, 3E10,15 ;21759          MOV WR[0] TO LS[T8]             ;FFFE DATA PATTERN IN LS TO WRITE
```

```
                    ;21760  LOOP.T35.9:
U 1C49, 99BA,35     ;21761          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                    ;21762                                          ;SET UP TO WRITE CONTROL REG 1 OF UBE
U 1C4A, 3210,15     ;21763          WRITE.MEM LS[T8]                ;WRITE FFFE(H) IN CONTROL REG 1 (DON'T
                    ;21764                                          ; SET GO BIT)
U 1C4B, 18BB,B5     ;21765          MEM.REQ[READ.P] ADRS[BECR1] DT[WORD]
                    ;21766                                          ;SET UP TO READ CONTROL REG 1 OF UBE
U 1C4C, 3022,15     ;21767          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CONTROL REG 1
U 1C4D, 3610,95     ;21768          MOV LS[T8] TO WR[1]             ;EXPECTED DATA (FFFE)
U 1C4E, 0869,3C     ;21769          JSR [CHECK.RESULT]              ;CHECK FOR CORRECT DATA
U 1C4F, 89C4,94     ;21770          JMP [LOOP.T35.9]                ;ERROR LOOP IF ENABLED
                    ;21771
U 1C50, FF82,15     ;21772          INC LS[ERROR.NUMBER]            ;ERROR A
U 1C51, 5F28,15     ;21773          MCOM LS[#FFFF] TO WR[0]         ;FFFF0000 IN WR
U 1C52, 3E8A,15     ;21774          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK BITS 0-15 ONLY
U 1C53, 19B8,35     ;21775          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                    ;21776                                          ;SET UP TO WRITE TO UBE ADDR REG
U 1C54, 329E,15     ;21777          WRITE.MEM LS[ONES]              ;WRITE ONES IN UBE ADDR REG
                    ;21778  LOOP.T35.A:
U 1C55, 0A1B,8C     ;21779          JSR [ISSUE.DCLO]                ;ISSUE UNIBUS DCLO TO UBE
U 1C56, 98B9,B5     ;21780          MEM.REQ[READ.P] ADRS[BEBA] DT[WORD]
                    ;21781                                          ;SET UP TO READ UBE ADDR REG
U 1C57, 3022,15     ;21782          MOV MEM.DATA TO WR[0]           ;READ CONTENTS OF UBE ADDR REG
U 1C58, 2F82,95     ;21783          CLR WR[1]                       ;EXPECTED DATA
U 1C59, 0869,3C     ;21784          JSR [CHECK.RESULT]              ;CHECK FOR UBE ADDR REG CLEARED
U 1C5A, 09C5,54     ;21785          JMP [LOOP.T35.A]                ;ERROR LOOP IF ENABLED
                    ;21786  END.T35:
```

```
                ;21787  .page   "  TEST 36 Verify Byte Access on Read to UNIBUS  (M8391 MCT Module)"
                ;21788  ;++
                ;21789  ;
                ;21790  ; Test Description:
                ;21791  ;
                ;21792  ;       This test is designed to check for faults in the control store PROMs
                ;21793  ;       by verifying all paths of the micro-code.  This test checks the path
                ;21794  ;       taken when a byte read is executed in the READ TO UNIBUS flow.  The
                ;21795  ;       test checks the byte read by writing the UBE data buffer (BEDB) with
                ;21796  ;       00FF(H) and reading each byte seperately.  This verifies the branch
                ;21797  ;       taken in the READ TO UNIBUS micro-flow when an odd address (LVA01 set)
                ;21798  ;       is accessed.
                ;21799  ;
                ;21800  ; Assumptions:
                ;21801  ;
                ;21802  ;       It is assumed that all previous tests have run successfully, and
                ;21803  ;       that all hardware (other than MCT control store PROMs) is operating
                ;21804  ;       correctly.
                ;21805  ;
                ;21806  ; Test Steps:
                ;21807  ;
                ;21808  ;       1) Set up error mask, error number, and module number in LS (for
                ;21809  ;          error printouts) and clear previous error number in LS.  Also
                ;21810  ;          issue DCLO to clear error bits in the UBE.
                ;21811  ;       2) Write 00FF in the UBE data buffer using a word data type.
                ;21812  ;       3) Issue a MEM.REQ data type byte to the low byte of the UBE data
                ;21813  ;          buffer and check results for all ones (sign extends).
                ;21814  ;       4) Issue a MEM.REQ data type byte to the high byte of the UBE data
                ;21815  ;          buffer and check results for all 0's.
                ;21816  ;
                ;21817  ; Errors:
                ;21818  ;
                ;21819  ;       Error 1 - Read byte from UBE failed on low byte (even address).
                ;21820  ;       Error 2 - Read byte from UBE failed on high byte (odd address).
                ;21821  ;
                ;21822  ; Debug:
                ;21823  ;
                ;21824  ;       Error 1 - This error is not likely to occur.  It is an indication of
                ;21825  ;       a problem with the memory data type logic.  Run previous modules
                ;21826  ;       again to try to pinpoint the failure.
                ;21827  ;
                ;21828  ;       Note:  The following error most likely indicates a problem with the
                ;21829  ;       micro-code flow.  The hardware used by the micro-code has been pre-
                ;21830  ;       viously tested.  To trace a problem, check the sequence outlined
                ;21831  ;       below.  If the sequence is OK, suspect a bad bit in one of the control
                ;21832  ;       store PROMs and check each state for the correct enables.
                ;21833  ;
                ;21834  ;       Error 2 - This error indicates a problem with the branch on LVA00.
                ;21835  ;       The flow should take the LVA00 = 1 path to the cycle that sets rotate
                ;21836  ;       byte and goes to ROTATE RIGHT C6.
                ;21837  ;
                ;21838  ;--
                ;21839
                ;21840  T.36:
U 1C5B, B65E,15 ;21841          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
```

```
                    ;21842                                  ; STATUS WORD                                                    U 1
U 1C5C, 3E80,15     ;21843          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD                   U 1
                    ;21844                                  ; BIT 15 INDICATES START OF TEST TO
                    ;21845                                  ; CONSOLE
U 1C5D, 10E0,15     ;21846          MISC [SET.CP.ATTN]      ;ISSUE CPU ATTN TO CONSOLE
                    ;21847  WAIT.T36.0:
U 1C5E, 89C5,E4     ;21848          JMP [WAIT.T36.0]        ;LOOP HERE WAITING FOR CONSOLE TO
                    ;21849                                  ; RESPOND
                    ;21850
U 1C5F, 0A1B,8C     ;21851          JSR [ISSUE.DCLO]        ;INITIALIZE UBE
U 1C60, 65A0,15     ;21852          CLR LS[PREVIOUS.ERROR]  ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 1C61, 5F28,15     ;21853          MCOM LS[#FFFF] TO WR[0] ;FFFF0000 IN WR
U 1C62, 3E8A,15     ;21854          MOV WR[0] TO LS[ERROR.MASK] ;SET UP ERROR MASK TO CHECK BITS 0-15
U 1C63, 3644,15     ;21855          MOV LS[MCT] TO WR[0]    ;SET UP MODULE CODE. BIT 2 SET IN WR
U 1C64, 3E8C,15     ;21856          MOV WR[0] TO LS[MODULE.NUM] ;STORE MODULE CODE IN LS TO INDICATE
                    ;21857                                  ; MCT BOARD
U 1C65, B640,15     ;21858          MOV LS[#1] TO WR[0]     ;1 IN WR
U 1C66, BE82,15     ;21859          MOV WR[0] TO LS[ERROR.NUMBER] ;SET UP ERROR NUMBER IN LS
                    ;21860
U 1C67, 19B4,35     ;21861          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                    ;21862                                  ;SET UP TO WRITE DATA BUFFER OF UBE
U 1C68, 3226,15     ;21863          WRITE.MEM LS[#FF]       ;WRITE 1'S IN LOW BYTE, 0'S IN HIGH
                    ;21864                                  ; BYTE OF DATA BUFFER
                    ;21865  LOOP.T36.1:
U 1C69, 2F80,15     ;21866          CLR WR[0]               ;ALL 0'S IN WR 0
U 1C6A, 18B5,95     ;21867          MEM.REQ[READ.P] ADRS[BEDB] DT[BYTE]
                    ;21868                                  ;SET UP TO READ LOW BYTE OF DATA BUFFER
U 1C6B, 3022,15     ;21869          MOV MEM.DATA TO WR[0]   ;GET CONTENTS OF DATA BUFFER
U 1C6C, 369E,95     ;21870          MOV LS[ONES] TO WR[1]   ;EXPECTED DATA (SIGN EXTENDS)
U 1C6D, 0869,3C     ;21871          JSR [CHECK.RESULT]      ;CHECK FOR ALL ONES
U 1C6E, 09C6,94     ;21872          JMP [LOOP.T36.1]        ;ERROR LOOP IF ENABLED
                    ;21873
U 1C6F, FF82,15     ;21874          INC LS[ERROR.NUMBER]    ;ERROR 2
U 1C70, 36B4,15     ;21875          MOV LS[BEDB] TO WR[0]   ;ADDRESS OF UBE DATA BUFFER
U 1C71, BE12,15     ;21876          MOV WR[0] TO LS[T9]      ;STORE ADDRESS IN LS
U 1C72, FF12,15     ;21877          INC LS[T9]              ;HIGH BYTE OF DATA BUFFER
                    ;21878  LOOP.T36.2:
U 1C73, B69E,15     ;21879          MOV LS[ONES] TO WR[0]   ;ONES IN WR 0
U 1C74, 1813,95     ;21880          MEM.REQ[READ.P] ADRS[T9] DT[BYTE]
                    ;21881                                  ;SET UP TO READ HIGH BYTE OF DATA BUFFER
U 1C75, 3022,15     ;21882          MOV MEM.DATA TO WR[0]   ;GET CONTENTS OF DATA BUFFER
U 1C76, B69C,95     ;21883          MOV LS[ZERO] TO WR[1]   ;EXPECTED DATA (SIGN EXTENDS)
U 1C77, 0869,3C     ;21884          JSR [CHECK.RESULT]      ;CHECK FOR ALL ZEROS
U 1C78, 89C7,34     ;21885          JMP [LOOP.T36.2]        ;ERROR LOOP IF ENABLED
                    ;21886
                    ;21887  END.T36:
```

```
;21888  .page   "  TEST 37 BR Interrupt Test  (M8391 MCT or M8390 CPU Module)"
;21889  ;++
;21890  ;
;21891  ; Test Description:
;21892  ;
;21893  ;       This test checks Bus Requests (BR's), Bus Grants (BG's) and passing
;21894  ;       of a vector on the D lines.  BR and BG levels 7-4 will be tested.
;21895  ;       The IPL will be set to a 0 to allow any BR interrupts through.  The
;21896  ;       UBE will be set up to issue a BR on level 7.  The CPU will wait for
;21897  ;       several cycles, then check for the BR interrupt.  If the BR 7 request
;21898  ;       is received correctly, a MEM.REQ is made to ISSUE.BG with the bits
;21899  ;       2 and 3 of the address bits set (LVA 02 and LVA 03 determine the BG
;21900  ;       level).
;21901  ;         ISSUE.BG performs the following:  After the dispatch, the next
;21902  ;       micro-cycle asserts ADDR PH, opens the gate to read the CP (to get the
;21903  ;       address bits 2 and 3), sets bypass and busy, and sets LOCK.  It branch-
;21904  ;       es on UB ACT and should take the UB ACT not path to ISS BG C4.  Errors
;21905  ;       are cleared and the BG on level 7 is issued.  The next cycle waits for
;21906  ;       the UBE to assert SACK on receiving the BG while keeping the UB DIR
;21907  ;       latch open and clocking the ERR registers.  It also checks for TIMOUT
;21908  ;       to be asserted.  At some point, either TIMEOUT or SACK will be received
;21909  ;       (depending on where we are in relation to when a refresh cycle occurs),
;21910  ;       and the micro-code will branch to the next cycle.
;21911  ;         If SACK comes before TIMEOUT, BG will be dropped and the next cycle
;21912  ;       will loop waiting for SACK to be deasserted by the UBE.  If TIMOUT
;21913  ;       comes before SACK, another path will be taken waiting for SACK to come
;21914  ;       up while keeping BG asserted.  After SACK is received, the micro-code
;21915  ;       waits for SACK to go away, and ends up at ISS BG C5A when SACK is
;21916  ;       dropped regardless of which branch it took at ISS BG C5.
;21917  ;         The UBE will have asserted INTR before dropping SACK, and ISS BG C5A
;21918  ;       clears LOCK and branches on INTR asserted.  The next cycle checks that
;21919  ;       INTR is still asserted, and branches to ISS BG C6.  This cycle closes
;21920  ;       the UB data latch, issues SSYN, and waits for INTR to drop.  The UBE
;21921  ;       will drop INTR on receiving SSYN.
;21922  ;         When INTR is dropped, the micro-code will branch to ISS BG C7.  This
;21923  ;       cycle drops MEM BSY and the CPU takes the data (which is the vector
;21924  ;       sent by the UBE).  When the CPU has the data (CPUG goes away), the
;21925  ;       cycle branches to IDLE.
;21926  ;         This sequence is repeated for BR levels 6, 5, and 4.  Each time the
;21927  ;       interrupt identifier code is checked, the NXM bit in CSR 1 is checked
;21928  ;       for 0, the UBE error register is checked for zeros, and the interrupt
;21929  ;       vector is checked (the vector is 148(H) in all cases).
;21930  ;
;21931  ; Assumptions:
;21932  ;
;21933  ;       It is assumed that all other micro-diagnostics have run successfully,
;21934  ;       in particular, the BUS IRQ TEST in ENKCB.
;21935  ;       It is also assumed that the UBE present is known to be good.
;21936  ;
;21937  ; Test Steps:
;21938  ;
;21939  ;       1)  Set up error mask, error number, and module number in LS (for
;21940  ;           error printouts) and clear previous error number in LS.  Also
;21941  ;           issue DCLO to UBE to clear error bits.
;21942  ;       2)  Clear IPL bits by clearing LS[PSL.HW].
```

```
;21943  ;       3)  Set up UBE cycle count reg for 1 cycle, and set control reg 1
;21944  ;           to issue a BR 7 and interrupt on becoming bus master.  Also
;21945  ;           set the GO bit which starts the UBE.
;21946  ;       4)  Delay for 2 cycles, then check for a BR interrupt.  If interrupt
;21947  ;           occurs check identifier code for a BR 7 interrupt.
;21948  ;       5)  Issue MEM.REQ [ISSUE.BG] using an address with bits 2 and 3 set.
;21949  ;           Issue MOV MEM.DATA TO LS[T8] to complete cycle and save interrupt
;21950  ;           vector.
;21951  ;       6)  Read CSR 1 and check for NXM clear.
;21952  ;       7)  Read UBE control reg 2 and check for 0 (no UBE error).
;21953  ;       8)  Check interrupt vector for a 148(H).
;21954  ;       9)  Repeat steps 3 through 7 with BR's on levels 6, 5, and 4.
;21955  ;
;21956  ; Errors:
;21957  ;
;21958  ;       NOTE:  Value under OTHER in error report is BR level being tested (7,
;21959  ;       6, 5, or 4).
;21960  ;
;21961  ;       Error 1 - No BR interrupt detected.
;21962  ;       Error 2 - BR interrupt occurred on wrong level.  Expected and received
;21963  ;                 is interrupt identifier code from the CPU.
;21964  ;       Error 3 - SACK timeout occured or INTR not received (NXM set in CSR 1).
;21965  ;       Error 4 - UBE error occurred.  Received data is contents of UBE control
;21966  ;                 reg 2.
;21967  ;       Error 5 - Interrupt vector incorrect or not received.  Expected and
;21968  ;                 received is the interrupt vector sent by the UBE after BG is
;21969  ;                 issued.
;21970  ;
;21971  ; Debug:
;21972  ;
;21973  ;       NOTE:  The MCT can hang if L INTR H is stuck high.  The hang would oc-
;21974  ;       cur at ISS BG C6 when the MCT micro-code is looping waiting for INTR
;21975  ;       to be deasserted.  If a monitor time out (24 error) occurs, check L
;21976  ;       INTR H first at the BEN 2 mux.
;21977  ;
;21978  ;       Error 1 - This error most likely indicates a problem with the BR lines
;21979  ;       on the CPU module as they leave the board, or possibly in the back-
;21980  ;       plane.  Check the BR line being tested while looping on error.  The
;21981  ;       logic from the receiver chip on to the interrupt logic has been tested
;21982  ;       previously.
;21983  ;
;21984  ;       Error 2 - This error is unlikely if the interrupt test in ENKCB has run
;21985  ;       successfully.  Suspect any new modules that have been added since run-
;21986  ;       ing that test or the UBE board itself.
;21987  ;
;21988  ;       Error 3 - This error indicates a problem with the BG logic, the SACK
;21989  ;       logic, or the INTR logic.  While looping on error, check UBS BGx H for
;21990  ;       a high on the BG that is being tested.  If BG is not correct, check
;21991  ;       the decoder chip.  ISSUE BG L should be pulsing low while LVA 2 and
;21992  ;       LVA 3 are selecting the proper grant.  If ISSUE BG L is not going low,
;21993  ;       check the output of the MISC CONTROL PAL.  If incorrect there check
;21994  ;       the inputs for a high on SPF0 H, SPF1 H, and SPF2 H during the SET BG
;21995  ;       micro-code at ISS BG C4 and ISS BG C5.  If correct, the PAL is probably
;21996  ;       bad.
;21997  ;         If the BG is OK, check that UBS SACK L is pulsing low during the
```

```
                ;21998  ;       loop and trace it through the latches to the BEN 3 branch mux.  If OK
                ;21999  ;       at the BEN 3 mux, suspect the ISSUE.BG micro-code itself.
                ;22000  ;         If SACK is ok, check that UBS INTR L is pulsing low during the loop.
                ;22001  ;       Trace it through the latches to the BEN 2 mux as L INTR H.  If it gets
                ;22002  ;       to the MUX OK, suspect the BEN 2 mux or the micro-code itself.
                ;22003  ;
                ;22004  ;       Error 4 - This error indicates that the UBE detected an error of some
                ;22005  ;       kind.  See the table of errors in the beginning of this listing to
                ;22006  ;       determine what went wrong.
                ;22007  ;
                ;22008  ;       Error 5 - This error is unlikely to come up by itself.  It indicates
                ;22009  ;       that the vector sent over the data lines by the UBE is incorrect.  The
                ;22010  ;       logic used to enable this data has been tested previously.  If this is
                ;22011  ;       the first error, suspect a micro-code problem at the cycle following
                ;22012  ;       ISS BC C5A, or the ISS BG C6 cycle.
                ;22013  ;
                ;22014  ;--
                ;22015
                ;22016  ;       T8  = RECEIVED INTERRUPT VECTOR FROM UBE
                ;22017  ;       T11 = DATA FOR UBE CONTROL REG 1
                ;22018  ;       T12 = EXPECTED BR IDENTIFIER CODE
                ;22019  ;       T13 = ADDRESS SENT WITH MEM.REG TO ISSUE BG (DETERMINES BG LEVEL)
                ;22020  ;       T15 = EXPECTED INTERRUPT VECTOR FROM UBE (148(H))
                ;22021
                ;22022  T.37:
U 1C79, B65E,15 ;22023          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;22024                                          ; STATUS WORD
U 1C7A, 3E80,15 ;22025          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;22026                                          ; BIT 15 INDICATES START OF TEST TO
                ;22027                                          ; CONSOLE
U 1C7B, 10E0,15 ;22028          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;22029  WAIT.T37.0:
U 1C7C, 89C7,C4 ;22030          JMP [WAIT.T37.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;22031                                          ; RESPOND
                ;22032
U 1C7D, 0A1B,8C ;22033          JSR [ISSUE.DCLO]                ;INIT UBE
U 1C7E, 65A0,15 ;22034          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 1C7F, DF29,15 ;22035          MCOM LS[#FFFF] TO WR[2]         ;FFFF0000 IN WR
U 1C80, 2F80,15 ;22036          CLR WR[0]                       ;CLEAR WR 0
U 1C81, BFFE,15 ;22037          MOV WR[0] TO LS[PSL.HW]         ;CLEAR IPL BITS IN PSW
U 1C82, 3711,95 ;22038          MOV LS[IDENT.MASK] TO WR[3]     ;ERROR MASK TO CHECK BITS 5-2 ONLY IN
                ;22039                                          ; BR IDENTIFIER
                ;22040
U 1C83, B670,15 ;22041          MOV LS[OTHER.DATA] TO WR[0]     ;SET BIT TO PRINT UNDER OTHER DATA
U 1C84, 3E80,15 ;22042          MOV WR[0] TO LS[CONTROL.STATUS] ;WRITE IN CONTROL AND STATUS WORD
U 1C85, 3650,15 ;22043          MOV LS[#100] TO WR[0]           ;100 IN WR
U 1C86, C74C,15 ;22044          BIS LS[#40] TO WR[0]            ;140 IN WR
U 1C87, C746,15 ;22045          BIS LS[#8] TO WR[0]             ;148(H) IN WR
U 1C88, BE1E,15 ;22046          MOV WR[0] TO LS[T15]            ;EXPECTED INTERRUPT VECTOR IN LS T15
                ;22047
U 1C89, 3648,15 ;22048          MOV LS[BR7] TO WR[0]            ;SET BIT 4 IN WR (UBE ISSUE BR7)
U 1C8A, C740,15 ;22049          BIS LS[GO] TO WR[0]             ;SET BIT 0 (UBE GO BIT)
U 1C8B, 3E16,15 ;22050          MOV WR[0] TO LS[T11]            ;SAVE IN LS T11
U 1C8C, B646,15 ;22051          MOV LS[#8] TO WR[0]             ;8 IN WR
U 1C8D, 2100,15 ;22052          DEC WR[0]                       ;7 IN WR
```

```
U 1C8E, BE88,15 ;22053          MOV WR[0] TO LS[ADDRESS.DATA]   ;7 IN OTHER DATA (BR7 BEING TESTED)
U 1C8F, 36CC,15 ;22054          MOV LS[BR7.IDENT] TO WR[0]      ;101000(B) IN WR
U 1C90, BE18,15 ;22055          MOV WR[0] TO LS[T12]            ;EXPECTED IDENTIFIER CODE FOR BR 7
                ;22056                                          ; (1010(B))
U 1C91, B6CE,15 ;22057          MOV LS[BG7] TO WR[0]            ;SET BIT 2 AND 3 IN WR
U 1C92, 3E1A,15 ;22058          MOV WR[0] TO LS[T13]            ;SET BITS 2 AND 3 AS ADDRESS TO ISSUE
                ;22059                                          ; BG 7
U 1C93, 89CB,2C ;22060          JSR [LOOP.T37.3.5]              ;PERFORM TEST ON BR 7
                ;22061
U 1C94, B646,15 ;22062          MOV LS[BR6] TO WR[0]            ;SET BIT 3 IN WR (UBE ISSUE BR6)
U 1C95, C740,15 ;22063          BIS LS[GO] TO WR[0]             ;SET BIT 0 (UBE GO BIT)
U 1C96, 3E16,15 ;22064          MOV WR[0] TO LS[T11]            ;SAVE IN LS T11
U 1C97, FC88,15 ;22065          DEC LS[ADDRESS.DATA]            ;6 IN OTHER DATA (BR6 BEING TESTED)
U 1C98, 3644,15 ;22066          MOV LS[BIT2] TO WR[0]           ;BIT 2 SET IN WR
U 1C99, ED18,15 ;22067          BIS WR[0] TO LS[T12]            ;EXPECTED IDENTIFIER CODE FOR BR 6
                ;22068                                          ; (1011(B))
U 1C9A, B646,15 ;22069          MOV LS[BG6] TO WR[0]            ;SET BIT 3 IN WR
U 1C9B, 3E1A,15 ;22070          MOV WR[0] TO LS[T13]            ;SET BITS 3 AS ADDRESS TO ISSUE BG 6
U 1C9C, 89CB,2C ;22071          JSR [LOOP.T37.3.5]              ;PERFORM TEST ON BR 6
                ;22072
U 1C9D, 3644,15 ;22073          MOV LS[BR5] TO WR[0]            ;SET BIT 2 IN WR (UBE ISSUE BR5)
U 1C9E, C740,15 ;22074          BIS LS[GO] TO WR[0]             ;SET BIT 0 (UBE GO BIT)
U 1C9F, 3E16,15 ;22075          MOV WR[0] TO LS[T11]            ;SAVE IN LS T11
U 1CA0, FC88,15 ;22076          DEC LS[ADDRESS.DATA]            ;5 IN OTHER DATA (BR5 BEING TESTED)
U 1CA1, 3648,15 ;22077          MOV LS[BIT4] TO WR[0]           ;BIT 4 SET IN WR
U 1CA2, C74A,15 ;22078          BIS LS[BIT5] TO WR[0]           ;ALSO SET BIT 5
U 1CA3, BE18,15 ;22079          MOV WR[0] TO LS[T12]            ;EXPECTED IDENTIFIER CODE FOR BR 5
                ;22080                                          ; (1100(B))
U 1CA4, 3644,15 ;22081          MOV LS[BIT2] TO WR[0]           ;SET BIT 2 IN WR
U 1CA5, 3E1A,15 ;22082          MOV WR[0] TO LS[T13]            ;SET BIT 2 AS ADDRESS TO ISSUE BG 5
U 1CA6, 89CB,2C ;22083          JSR [LOOP.T37.3.5]              ;PERFORM TEST ON BR 5
                ;22084
U 1CA7, 3642,15 ;22085          MOV LS[BR4] TO WR[0]            ;SET BIT 1 IN WR (UBE ISSUE BR4)
U 1CA8, C740,15 ;22086          BIS LS[GO] TO WR[0]             ;SET BIT 0 (UBE GO BIT)
U 1CA9, 3E16,15 ;22087          MOV WR[0] TO LS[T11]            ;SAVE IN LS T11
U 1CAA, FC88,15 ;22088          DEC LS[ADDRESS.DATA]            ;4 IN OTHER DATA (BR4 BEING TESTED)
U 1CAB, B646,15 ;22089          MOV LS[BIT3] TO WR[0]           ;BIT 3 SET IN WR
U 1CAC, ED18,15 ;22090          BIS WR[0] TO LS[T12]            ;EXPECTED IDENTIFIER CODE FOR BR 4
                ;22091                                          ; (1110(B))
U 1CAD, E51A,15 ;22092          CLR LS[T13]                     ;ADDRESS TO ISSUE BG 4
U 1CAE, 89CB,2C ;22093          JSR [LOOP.T37.3.5]              ;PERFORM TEST ON BR 4
                ;22094
U 1CAF, 09CE,74 ;22095          JMP [END.T37]                   ;DONE
                ;22096
                ;22097  ;
                ;22098  ;  THIS SUBROUTINE DOES ALL THE CHECKING FOR EACH BR LEVEL.  VARIOUS LOCATIONS
                ;22099  ;  ARE SET UP IN LS BEFORE ENTRY AS EXPECTED DATA OR CONTROL FUNCTIONS.
                ;22100  ;
                ;22101
                ;22102  LOOP.T37.1.2:
U 1CB0, 9F1A,35 ;22103          MEM.REQ[ISSUE.BG] ADRS[T13] DT[WORD]
                ;22104                                          ;ISSUE BG ON CURRENT LEVEL
U 1CB1, 3810,15 ;22105          MOV MEM.DATA TO LS[T8]          ;GET INTERRUPT VECTOR SENT BY UBE
                ;22106  LOOP.T37.3.5:
U 1CB2, 3642,15 ;22107          MOV LS[CPU] TO WR[0]            ;INDICATE CPU BOARD.
```

```
U 1CB3, 3E8C,15 ;22108          MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                ;22109                                          ; CPU BOARD.
U 1CB4, B640,15 ;22110          MOV LS[#1] TO WR[0]             ;1 IN WR
U 1CB5, BE82,15 ;22111          MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
U 1CB6, 99B6,35 ;22112          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                ;22113                                          ;SET UP TO WRITE TO UBE CYCLE COUNT
U 1CB7, 329E,15 ;22114          WRITE.MEM LS[ONES]              ;PUT A -1 IN CYCLE COUNT (DO 1 CYCLE)
U 1CB8, 99BA,35 ;22115          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                ;22116                                          ;SET UP TO WRITE TO UBE CONTROL REG 1
U 1CB9, 3216,15 ;22117          WRITE.MEM LS[T11]               ;SET UP CONTROL REG TO ISSUE BR AND
                ;22118                                          ; INTERRUPT ON BECOMING BUS MASTER
U 1CBA, DB00,15 ;22119          NOP                             ;WAIT FOR UBE TO ASSERT BR
U 1CBB, DB00,15 ;22120          NOP                             ;WAIT FOR UBE TO ASSERT BR
U 1CBC, DB00,15 ;22121          NOP                             ;WAIT FOR BR TO LATCH INTO INTERRUPT
U 1CBD, DB00,15 ;22122          NOP                             ;WAIT FOR BR TO LATCH INTO INTERRUPT
U 1CBE, 09CD,E7 ;22123          JMP.IF[NO.INTERRUPT] TO [T37.ERROR.1]  ;ERROR IF NO INTERRUPT
                ;22124  T37.2:
U 1CBF, FF82,15 ;22125          INC LS[ERROR.NUMBER]            ;ERROR 2
U 1CC0, 3E8B,95 ;22126          MOV WR[3] TO LS[ERROR.MASK]     ;CHECK BITS 5-2 ONLY
U 1CC1, 36FC,15 ;22127          MOV LS[INTERRUPT.VEC] TO WR[0]  ;GET IDENTIFIER CODE
U 1CC2, B618,95 ;22128          MOV LS[T12] TO WR[1]            ;EXPECTED IDENTIFIER CODE FOR BR
U 1CC3, 0869,3C ;22129          JSR [CHECK.RESULT]              ;CHECK FOR CORRECT IDENTIFIER CODE
U 1CC4, 89CB,04 ;22130          JMP [LOOP.T37.1.2]              ;ERROR LOOP IF ENABLED
                ;22131
U 1CC5, FF82,15 ;22132          INC LS[ERROR.NUMBER]            ;ERROR 3
U 1CC6, 3644,15 ;22133          MOV LS[MCT] TO WR[0]            ;CALL OUT MCT MODULE
U 1CC7, 3E8C,15 ;22134          MOV WR[0] TO LS[MODULE.NUM]     ;STORE MODULE CODE IN LS TO INDICATE
                ;22135                                          ; MCT BOARD.
U 1CC8, 5F60,15 ;22136          MCOM LS[NXM] TO WR[0]           ;CLEAR BIT 16 IN WR 0
U 1CC9, 3E8A,15 ;22137          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK BIT 16 (NXM IN CSR 1) ONLY
U 1CCA, 9F1A,35 ;22138          MEM.REQ[ISSUE.BG] ADRS[T13] DT[WORD]
                ;22139                                          ;ISSUE BG ON CURRENT LEVEL
U 1CCB, 3810,15 ;22140          MOV MEM.DATA TO LS[T8]          ;GET INTERRUPT VECTOR SENT BY UBE
U 1CCC, 9D45,75 ;22141          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;22142                                          ;SET UP TO READ CSR 1
U 1CCD, 3022,15 ;22143          MOV MEM.DATA TO WR[0]           ;READ CSR 1
U 1CCE, 2F82,95 ;22144          CLR WR[1]                       ;EXPECTED DATA (NXM CLEAR)
U 1CCF, 0869,3C ;22145          JSR [CHECK.RESULT]              ;CHECK FOR NXM CLEAR
U 1CD0, 09CB,24 ;22146          JMP [LOOP.T37.3.5]              ;ERROR LOOP IF ENABLED
                ;22147
U 1CD1, FF82,15 ;22148          INC LS[ERROR.NUMBER]            ;ERROR 4
U 1CD2, BE8B,15 ;22149          MOV WR[2] TO LS[ERROR.MASK]     ;CHECK BITS 15-00
U 1CD3, 98BF,B5 ;22150          MEM.REQ[READ.P] ADRS[BECR2] DT[WORD]
                ;22151                                          ;READ UBE CONTROL REG 2
U 1CD4, 3022,15 ;22152          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CONTROL REG 2
U 1CD5, B65A,95 ;22153          MOV LS[CCOVF] TO WR[1]          ;EXPECTED DATA (CCOVF SET, ERROR BITS
                ;22154                                          ; CLEAR)
U 1CD6, 0869,3C ;22155          JSR [CHECK.RESULT]              ;CHECK FOR ERRORS
U 1CD7, 09CB,24 ;22156          JMP [LOOP.T37.3.5]              ;ERROR LOOP IF ENABLED
                ;22157
U 1CD8, FF82,15 ;22158          INC LS[ERROR.NUMBER]            ;ERROR 5
U 1CD9, B610,15 ;22159          MOV LS[T8] TO WR[0]             ;INTERRUPT VECTOR RECEIVED
U 1CDA, B61E,95 ;22160          MOV LS[T15] TO WR[1]            ;INTERRUPT VECTOR EXPECTED
U 1CDB, 0869,3C ;22161          JSR [CHECK.RESULT]              ;SEE IF VECTOR = 148(H)
U 1CDC, 09CB,24 ;22162          JMP [LOOP.T37.3.5]              ;ERROR LOOP IF ENABLED
```

```
U 1CDD, 5B00,14 ;22163          RETURN                              ;DONE
                ;22164
                ;22165
                ;22166  T37.ERROR.1:
U 1CDE, B66E,15 ;22167          MOV LS[NA.EXP.REC] TO WR[0]         ;SET BIT TO PRINT N/A UNDER EXP AND REC
U 1CDF, 6D80,15 ;22168          BIS WR[0] TO LS[CONTROL.STATUS]     ;WRITE IN CONTROL AND STATUS WORD. KEEP
                ;22169                                              ; 'OTHER' BIT SET TO PRINT UNDER OTHER
U 1CE0, 2F82,95 ;22170          CLR WR[1]                           ;FAKE 0'S EXPECTED DATA
U 1CE1, B69E,15 ;22171          MOV LS[ONES] TO WR[0]               ;FAKE 1'S RECEIVED DATA
U 1CE2, 0869,3C ;22172          JSR [CHECK.RESULT]                  ;CALL ERROR ROUTINE
U 1CE3, 89CB,04 ;22173          JMP [LOOP.T37.1.2]                  ;ERROR LOOP IF ENABLED
U 1CE4, B670,15 ;22174          MOV LS[OTHER.DATA] TO WR[0]         ;SET UP TO PRINT UNDER OTHER DATA
U 1CE5, 3E80,15 ;22175          MOV WR[0] TO LS[CONTROL.STATUS]     ;WRITE IN CONTROL AND STATUS WORD
U 1CE6, 89CB,F4 ;22176          JMP [T37.2]                         ;CONTINUE WITH TEST
                ;22177  END.T37:
```

```
;22178  .page   "  TEST 38 UBE Write To Array Test  (M8391 MCT Module)"
;22179  ;++
;22180  ;
;22181  ; Test Description:
;22182  ;
;22183  ;       This test checks the logic and micro-flow associated with the UNIBUS
;22184  ;       WRITE TO ARRAY routine.  The is the first test to exit the IDLE loop
;22185  ;       through the MSYN branch path.
;22186  ;         The test sets up the UBE to do a WRITE TO ARRAY.  The UBE first as-
;22187  ;       serts NPR to the MCT, and the MCT responds with NPG.  The UBE will as-
;22188  ;       sert SACK when it sees NPG.  The MCT will then drop NPG.  The UBE will
;22189  ;       assert BBSY and negate SACK.  The UBE can now issue MSYN and begin the
;22190  ;       transfer cycle.
;22191  ;         Now the data path is as follows.  After the UBE issues MSYN,  the MCT
;22192  ;       exits from the idle loop taking the MSYN path to UB CYC C2.  The UB
;22193  ;       area of the TB RAMS is selected and REF and VALID are checked.  The
;22194  ;       microcode takes the VALID path to UB CYC C3A (VALID comes from the
;22195  ;       UNIBUS MAP area of the TB RAMS).  At UB CYC C3A the data latches for
;22196  ;       UB data are opened and the direction set up in preperation for a write
;22197  ;       to array.  A branch on UBC1 and MSYN is performed.  The microcode takes
;22198  ;       the MSYN and UBC1 path (UBC1 is high since this is a DATO from the UBE)
;22199  ;       to the UNIBUS WRITE TO ARRAY flows.  Here the ECC latches are opened, a
;22200  ;       CSR ERR SUM CLK is asserted, and a branch on PAGE BOUND and TWO MEM
;22201  ;       CYCLES is taken.
;22202  ;         The PAGE BOUND NOT and TWO MEM CYCLE NOT path is taken to a micro
;22203  ;       cycle which does a READ ARRAY and branches on UB ERRSUM not to the next
;22204  ;       cycle.  This cycle issues SSYN, sets DATI, and sets up to write the
;22205  ;       incoming data word.  It then goes to UB CYC WR ARY C7 to continue the
;22206  ;       write and branch on ERR.  The ERR not branch is taken to UB CYC WR ARY
;22207  ;       C8 where WRITE TIM to the array is started.  The 1 MEM CYC path is
;22208  ;       taken here.  The next three cycle complete the write.
;22209  ;         After the UBE receives the SSYN issued by the MCT, the UBE will drop
;22210  ;       MSYN.  When the MCT receives the negation of MSYN, it will drop SSYN.
;22211  ;       The UBE will also drop BBSY, after a short delay, when it receives
;22212  ;       SSYN.  The MCT will deassert UB ACTIVITY when it sees BBSY go away.
;22213  ;         The last of the three cycle branches on UB ACT and SSYN.  Since UB
;22214  ;       ACT and SSYN have both been dropped by the MCT according to the pre-
;22215  ;       ceeding paragraph, the branch to UB CYC WR ARY DONE is taken.  The MCT
;22216  ;       micro flow then goes to IDLE.
;22217  ;         Data patterns used are all 1's and all 0's.
;22218  ;
;22219  ; Assumptions:
;22220  ;
;22221  ;       It is assumed that all other micro-diagnostics have run successfully.
;22222  ;       It is also assumed that the UBE present is known to be good.
;22223  ;
;22224  ; Test Steps:
;22225  ;
;22226  ;       1)  Set up error mask, error number, and module number in LS (for
;22227  ;           error printouts) and clear previous error number in LS.  Also
;22228  ;           issue DCLO to UBE to clear error bits.
;22229  ;       2)  Set up UNIBUS MAP to map virtual address 0 to physical address 0
;22230  ;           and set the VALID bit.
;22231  ;       3)  Clear the IPL bits in the PSW (allow all interrupts).  Write all
;22232  ;           1's at physical memory address 0.
```

```
;22233  ;       4)  Set up UBE data reg with all 0's, cycle count to 1 cycle (FFFF),
;22234  ;           and address reg to address 0.
;22235  ;       5)  Set up control reg 1 of the UBE to execute one DATO cycle via an
;22236  ;           NPR and interrupt via a BR6 when done.  Also set the GO bit to
;22237  ;           start the UBE.
;22238  ;       6)  Execute a delay loop to allow time for BR interrupt to occur (due
;22239  ;           to either a UBE error or done).  Check that BR interrupt occurs.
;22240  ;       7)  Check the identifier code if BR interrupt occurs.  If a BR 7 was
;22241  ;           received (indicates a UBE error), read the UBE error register and
;22242  ;           report the error.
;22243  ;       8)  Read memory address 0 and check for all 0's.
;22244  ;       9)  Read CSR 2 (UNIBUS CSR) and check for 0's in error bits.  This
;22245  ;           check is mainly to help in debug if the UB cycle is aborted.
;22246  ;       10) Write address 0 with longword of zeros and repeat steps 4 through
;22247  ;           9 with all 1's as data.
;22248  ;       11) Change the address reg to write at address 2 and repeat steps 4
;22249  ;           through 9 with all 1's as data.
;22250  ;       12) Set the BYTE OFFSET bit in the UB MAP and repeat steps 4 through 9
;22251  ;           with all 1's as data.
;22252  ;
;22253  ; Errors:
;22254  ;
;22255  ;       Note:  Data under OTHER in error report is physical addres. that the
;22256  ;       UBE is trying to write to in main memory.  The debug section will
;22257  ;       descibe the differences for error 3 when different portions of the
;22258  ;       longword are written.
;22259  ;
;22260  ;       Error 1 - No BR interrupt occurred.  May indicate NPR/NPG failure.
;22261  ;       Error 2 - BR7 interrupt occurred (indicates UBE error).  Received data
;22262  ;                 is contents of UBE control reg 2 (error register).
;22263  ;       Error 3 - Main memory longword 0 failed to get written correctly by UBE.
;22264  ;       Error 4 - CSR 2 (UB error CSR) had one or more error bits set.
;22265  ;
;22266  ; Debug:
;22267  ;
;22268  ;       Error 1 - This error indicates that the BR interrupt expected when the
;22269  ;       UBE finished its transfer, or upon a UBE error did not occur. The logic
;22270  ;       associated with NPR and NPG may be failing.  Set a SOMM at the CPU in-
;22271  ;       struction that jumps to the error if no interrupt. Check the signal UBS
;22272  ;       NPR L for a low.  Trace it through the latches to L NPR H, where it
;22273  ;       should be high, to the ARBITRATOR PAL. If it is high at the ARBITRATOR,
;22274  ;       check the other inputs for a low on ARB C0 H and L SACK H, and a high
;22275  ;       on LOCK L (formerly IGNORE MSYN L).  If the inputs are OK, check for a
;22276  ;       low on the output NPG L.  If this is incorrect, the PAL is probably
;22277  ;       bad.  If NPG L is low, trace it out to the finger where it is UBS NPG
;22278  ;       H.
;22279  ;
;22280  ;       Error 2 - This error indicates that the UBE detected an error.  See the
;22281  ;       UBE error register list in this listing to determine what went wrong.
;22282  ;       Also see debug for error 3 below.
;22283  ;
;22284  ;       Error 3 - This error indicates that the UBE failed to write data cor-
;22285  ;       rectly in the array.  If the cycle was aborted, there could be a prob-
;22286  ;       lem with the UB ERR SUM logic.  This error would have also set bit 8
;22287  ;       (No SSYN Err) in the UBE control reg 2.  While looping on error, check
```

```
;22288  ;       UB ERR SUM L out of the UB CSR 2 PAL.  It should be high during the
;22289  ;       READ ARRAY cycle that branches on UB ERRSUM.  If not check the inputs
;22290  ;       to the PAL for a high on NXM L, WR NOT VAILD L, and VALID H.  If these
;22291  ;       inputs are OK, then the PAL is probably bad.  WR NOT VALID comes from
;22292  ;       the CSR 1B PAL.  If it is low, check the inputs for a high on 2 MEM
;22293  ;       CYCLES L.  If this is OK, the CSR 1B PAL is probably bad.  The other
;22294  ;       inputs to the UB CSR 2 PAL have been tested previously at their source.
;22295  ;         If UB ERRSUM is OK, check that it gets to the BEN 3 MUX correctly.
;22296  ;       If the micro-code takes the UB ERRSUM path, the MUX may be bad.
;22297  ;
;22298  ;       OTHER=0
;22299  ;
;22300  ;         If the result is shifted or only one byte is written, suspect the
;22301  ;       DATA ROTATOR CONTROL PAL or the MDR AND DIR CONTROL PAL.  The inputs
;22302  ;       to the MDR AND DIR CONTROL PAL during the WRITE SUBSTITUTE cycles
;22303  ;       should be as follows: RS1 L should be high, RS0 L should be low, A0 L
;22304  ;       and A1 L should both be high, ROT C0 H should be high, CPH/UBL should
;22305  ;       be low, and 2ND MEM CYC H should be low.  The output should be low to
;22306  ;       the input of LAT OUTPUT BYT 2 L and LAT OUTPUT BYT 3 L. The output
;22307  ;       should be high to the input of LAT OUTPUT BYT 0 L and LAT OUTPUT BYT 1
;22308  ;       L.  If the inputs are ok but the output are wrong, the PAL is probably
;22309  ;       bad.
;22310  ;         If the result is shifted or only one byte is written, and the inputs
;22311  ;       to the MDR AND DIR CONTROL PAL from the DATA ROTATOR PAL are not as
;22312  ;       descibed above, check the DATA ROTATOR CONTROL PAL for the following
;22313  ;       inputs during the cycle that asserts ROT CLOCK H (UB CYC C3A):  CPH/UBL
;22314  ;       should be low, BYTE OFFSET H should be low, LVA 01 H should be low, and
;22315  ;       ROT C0 H should be low.  If these inputs are OK, the PAL is probably
;22316  ;       bad.
;22317  ;
;22318  ;       OTHER=2
;22319  ;
;22320  ;         The difference between writing at byte 2 from writing at byte 0 is
;22321  ;       that LVA 01 H going into the DATA ROTATOR PAL is high.  The other in-
;22322  ;       puts are as above.  The only output change on the DATA ROTATOR PAL is
;22323  ;       that A1 L is low out of the DATA ROTATOR PAL.  This switches the output
;22324  ;       of the MDR AND DIR CONTROL PAL so that the input to LAT OUTPUT BYT 0 L
;22325  ;       and LATCH OUTPUT BYTE 1 L is low, while the other two bytes have a high
;22326  ;       input.
;22327  ;
;22328  ;       OTHER=1
;22329  ;
;22330  ;         The difference between writing at byte 1 from writing at byte 0 is
;22331  ;       that BYTE OFFSET H is high into the DATA ROTATOR PAL.  The other in-
;22332  ;       puts are as in OTHER=0 above.  The only output of the DATA ROTATOR PAL
;22333  ;       that is different is that A0 L is low.  This inputs into the MDR AND
;22334  ;       DIR CONTROL PAL to produce a low at the inputs to LAT OUTPUT BYT 0 L
;22335  ;       and LAT OUTPUT BYT 3 L, while the other 2 bytes have a high input.
;22336  ;
;22337  ;       Error 4 - If this is the first error, it indicates a problem with the
;22338  ;       UB CSR 2 PAL or its inputs.  While looping on error, check that the
;22339  ;       input CLR ERR L is going low at the time that CSR 2 CLK H is high at
;22340  ;       some time in the loop.  This combination should force the register
;22341  ;       outputs low.  If not, the PAL is bad.
;22342  ;         If the outputs are OK, check that the input RD CSR 2 L is pulsing
```

```
                ;22343  ;       low.  The PAL is bad if RD CSR 2 L goes low.  If RD CSR 2 L is always
                ;22344  ;       high, trace it back through the decoder to find the failure.
                ;22345  ;
                ;22346  ;--
                ;22347
                ;22348  ;       T14 = DATA FOR UBE CONTROL REG 1
                ;22349  ;       T15 = EXPECTED ARRAY DATA
                ;22350  ;       T57 = ADDRESS TO ISSUE BG 7
                ;22351  ;       T58 = ADDRESS TO WRITE IN UBE ADDRESS REG
                ;22352
                ;22353  T.38:
U 1CE7, B65E,15 ;22354          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;22355                                          ; STATUS WORD
U 1CE8, 3E80,15 ;22356          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;22357                                          ; BIT 15 INDICATES START OF TEST TO
                ;22358                                          ; CONSOLE
U 1CE9, 10E0,15 ;22359          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;22360  WAIT.T38.0:
U 1CEA, 89CE,A4 ;22361          JMP [WAIT.T38.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;22362                                          ; RESPOND
                ;22363
U 1CEB, 0A1B,8C ;22364          JSR [ISSUE.DCLO]                ;INIT UBE
U 1CEC, 65A0,15 ;22365          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 1CED, 3644,15 ;22366          MOV LS[MCT] TO WR[0]            ;SET BIT 2 IN WR
U 1CEE, 3E8C,15 ;22367          MOV WR[0] TO LS[MODULE.NUM]     ;SET UP TO CALL OUT MCT MODULE IF ERROR
U 1CEF, 1B9D,F5 ;22368          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;22369                                          ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1CF0, B27E,15 ;22370          WRITE.MEM LS[BIT31]             ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;22371                                          ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                ;22372                                          ; OFFSET CLEAR
U 1CF1, 2F80,15 ;22373          CLR WR[0]                       ;CLEAR WR 0
U 1CF2, BFFE,15 ;22374          MOV WR[0] TO LS[PSL.HW]         ;CLEAR IPL BITS IN PSW
U 1CF3, 999C,75 ;22375          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;22376                                          ;SET UP TO WRITE AT MEMORY ADDR 0
U 1CF4, 329E,15 ;22377          WRITE.MEM LS[ONES]              ;WRITE ALL 1'S AT ADDR 0
                ;22378
U 1CF5, 0A1C,0C ;22379          JSR [SETUP.DATO.NPR]            ;SETUP T14 WITH UBE DATA TO ISSUE NPR
                ;22380                                          ; DATO, INTERRUPT ON DONE AT BR6
                ;22381
U 1CF6, B670,15 ;22382          MOV LS[OTHER.DATA] TO WR[0]     ;SET BIT TO PRINT UNDER OTHER DATA
U 1CF7, 3E80,15 ;22383          MOV WR[0] TO LS[CONTROL.STATUS] ;WRITE IN CONTROL AND STATUS WORD
U 1CF8, 6588,15 ;22384          CLR LS[ADDRESS.DATA]            ;0 IN OTHER DATA (ADDRESS BEING TESTED)
U 1CF9, E5B0,15 ;22385          CLR LS[T58]                     ;CLEAR DATA TO WRITE IN UBE ADDRESS REG
                ;22386
U 1CFA, 19B4,35 ;22387          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                ;22388                                          ;SET UP TO WRITE TO UBE DATA REG
U 1CFB, B29C,15 ;22389          WRITE.MEM LS[ZERO]              ;PUT A 0 IN DATA REG
U 1CFC, 5F28,15 ;22390          MCOM LS[#FFFF] TO WR[0]         ;FFFF0000 IN WR
U 1CFD, BE1E,15 ;22391          MOV WR[0] TO LS[T15]            ;EXPECTED DATA
U 1CFE, 09D1,DC ;22392          JSR [LOOP.T38.4]                ;PERFORM TEST WITH ALL 0'S DATA
                ;22393                                          ; TO ADDRESS 0
                ;22394
U 1CFF, 999C,75 ;22395          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;22396                                          ;SET UP TO WRITE ARRAY ADDRESS 0
U 1D00, B29C,15 ;22397          WRITE.MEM LS[ZERO]              ;WRITE LONGWORD OF 0'S AT ADDRESS 0
```

```
U 1D01, 19B4,35 ;22398          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                ;22399                                  ;SET UP TO WRITE TO UBE DATA REG
U 1D02, 329E,15 ;22400          WRITE.MEM LS[ONES]      ;PUT ALL 1'S IN DATA REG
U 1D03, 3628,15 ;22401          MOV LS[#FFFF] TO WR[0]  ;0000FFFF IN WR
U 1D04, BE1E,15 ;22402          MOV WR[0] TO LS[T15]    ;EXPECTED DATA
U 1D05, 09D1,DC ;22403          JSR [LOOP.T38.4]        ;PERFORM TEST WITH ALL 1'S DATA
                ;22404                                  ; TO ADDRESS 0
                ;22405
U 1D06, 3642,15 ;22406          MOV LS[#2] TO WR[0]     ;2 IN WR
U 1D07, BE88,15 ;22407          MOV WR[0] TO LS[ADDRESS.DATA]   ;OTHER DATA IS 2 (ADDRESS 2)
U 1D08, 3EB0,15 ;22408          MOV WR[0] TO LS[T58]    ;DATA OF 2 TO WRITE IN UBE ADDRESS REG
U 1D09, 999C,75 ;22409          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;22410                                  ;SET UP TO WRITE ARRAY ADDRESS 0
U 1D0A, B29C,15 ;22411          WRITE.MEM LS[ZERO]      ;WRITE LONGWORD OF 0'S AT ADDRESS 0
U 1D0B, 5F28,15 ;22412          MCOM LS[#FFFF] TO WR[0] ;FFFF0000 IN WR
U 1D0C, BE1E,15 ;22413          MOV WR[0] TO LS[T15]    ;EXPECTED DATA
U 1D0D, 09D1,DC ;22414          JSR [LOOP.T38.4]        ;PERFORM TEST WITH ALL 1'S DATA
                ;22415                                  ; TO ADDRESS 2
                ;22416
U 1D0E, 367E,15 ;22417          MOV LS[BIT31] TO WR[0]  ;SET BIT 31 IN WR (VALID BIT)
U 1D0F, 4772,15 ;22418          BIS LS[BIT25] TO WR[0]  ;SET BIT 25 IN WR (BYTE OFFSET)
U 1D10, BEB2,15 ;22419          MOV WR[0] TO LS[T59]    ;STORE IN LS
U 1D11, 1B9D,F5 ;22420          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;22421                                  ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1D12, B2B2,15 ;22422          WRITE.MEM LS[T59]       ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;22423                                  ; ADDR 0 TO PHYSICAL ADDR 0, SET BYTE
                ;22424                                  ; OFFSET BIT
                ;22425
U 1D13, FC88,15 ;22426          DEC LS[ADDRESS.DATA]    ;OTHER DATA IS 1 (ADDRESS 1)
U 1D14, E5B0,15 ;22427          CLR LS[T58]             ;DATA OF 0 TO WRITE IN UBE ADDRESS REG
U 1D15, 999C,75 ;22428          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;22429                                  ;SET UP TO WRITE ARRAY ADDRESS 0
U 1D16, B29C,15 ;22430          WRITE.MEM LS[ZERO]      ;WRITE LONGWORD OF 0'S AT ADDRESS 0
U 1D17, B62C,15 ;22431          MOV LS[#FFFFFF00] TO WR[0]      ;FFFFFF00 IN WR
U 1D18, 452A,15 ;22432          BIC LS[#FF000000] TO WR[0]      ;00FFFF00 IN WR
U 1D19, BE1E,15 ;22433          MOV WR[0] TO LS[T15]    ;EXPECTED DATA
U 1D1A, 09D1,DC ;22434          JSR [LOOP.T38.4]        ;PERFORM TEST WITH ALL 1'S DATA
                ;22435                                  ; TO ADDRESS 1
 U 1D1B, 09D4,A4 ;22436         JMP [END.T38]           ;DONE
                ;22437
                ;22438 ;
                ;22439 ;SUBROUTINE TO PERFORM NPR TEST
                ;22440 ;
                ;22441 LOOP.T38.1:
U 1D1C, 0A1B,8C ;22442          JSR [ISSUE.DCLO]        ;INIT UBE FOR ERROR LOOP ON ERROR 1
                ;22443 LOOP.T38.4:
U 1D1D, 9D47,F5 ;22444          MEM.REQ[WRITE.CSR] ADRS[CSR2] DT[LONG]
                ;22445                                  ;SET UP TO WRITE UB CSR TO CLEAR ERRORS
U 1D1E, B29C,15 ;22446          WRITE.MEM LS[ZERO]      ;CLEAR CSR 2 ERROR BITS
                ;22447 LOOP.T38.2.3:
U 1D1F, B640,15 ;22448          MOV LS[#1] TO WR[0]     ;1 IN WR
U 1D20, BE82,15 ;22449          MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
                ;22450
 U 1D21, 19B8,35 ;22451         MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                ;22452                                  ;SET UP TO WRITE TO UBE ADDRESS REG
```

```
U 1D22, 32B0,15 ;22453          WRITE.MEM LS[T58]                ;PUT A 0 OR 2 IN ADDRESS REG
U 1D23, 99B6,35 ;22454          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                ;22455                                           ;SET UP TO WRITE TO UBE CYCLE COUNT
U 1D24, 329E,15 ;22456          WRITE.MEM LS[ONES]               ;PUT A -1 IN CYCLE COUNT (DO 1 CYCLE)
U 1D25, 99BA,35 ;22457          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                ;22458                                           ;SET UP TO WRITE TO UBE CONTROL REG 1
U 1D26, 321C,15 ;22459          WRITE.MEM LS[T14]                ;SET UP CONTROL REG TO ISSUE NPR TO
                ;22460                                           ; WRITE ARRAY AND INTR AT BR6 ON DONE
U 1D27, 364B,15 ;22461          MOV LS[#20] TO WR[2]             ;COUNTER TO WAIT FOR UBE TO FINISH
                ;22462  LOOP.T38.INT:
U 1D28, DB00,15 ;22463          NOP                              ;WAIT FOR UBE TO FINISH
                ;22464          DEC WR[2],                       ;DECREMENT COUNTER
U 1D29, A105,35 ;22465           DT(LONG)&SET.ALU.CC             ; AND SET CONDITION CODES
U 1D2A, 89D2,81 ;22466          JMP.IF[NEQ] TO [LOOP.T38.INT]    ;LOOP UNTIL COUNTER=0
U 1D2B, 89D4,17 ;22467          JMP.IF[NO.INTERRUPT] TO [T38.ERROR.1] ;REPORT ERROR IF NO BR INTERRUPT
                ;22468
                ;22469  T38.2:
U 1D2C, FF82,15 ;22470          INC LS[ERROR.NUMBER]             ;ERROR 2
U 1D2D, 8A1C,DC ;22471          JSR [CHECK.UBE.ERR]              ;SEE IF UBE ERR AND ISSUE NECESSARY
                ;22472                                           ; GRANTS
U 1D2E, 09D1,F4 ;22473          JMP [LOOP.T38.2.3]               ;ERROR LOOP RETURN IF ENABLED
                ;22474
U 1D2F, FF82,15 ;22475          INC LS[ERROR.NUMBER]             ;ERROR 3
U 1D30, E58A,15 ;22476          CLR LS[ERROR.MASK]               ;CHECK ALL BITS
U 1D31, 189D,F5 ;22477          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;22478                                           ;SET UP TO READ DATA AT ADDRESS 0
U 1D32, 3022,15 ;22479          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 1D33, B61E,95 ;22480          MOV LS[T15] TO WR[1]             ;EXPECTED DATA
U 1D34, 0869,3C ;22481          JSR [CHECK.RESULT]               ;SEE IF ARRAY WRITTEN CORRECTLY
U 1D35, 09D1,F4 ;22482          JMP [LOOP.T38.2.3]               ;ERROR LOOP IF ENABLED
                ;22483
U 1D36, FF82,15 ;22484          INC LS[ERROR.NUMBER]             ;ERROR 4
U 1D37, 3630,15 ;22485          MOV LS[CSR2.MASK] TO WR[0]       ;CLEAR BITS 16-14 AND 31 IN WR
U 1D38, 3E8A,15 ;22486          MOV WR[0] TO LS[ERROR.MASK]      ;CHECK ERROR BITS OF CSR 2 ONLY
U 1D39, 1D47,75 ;22487          MEM.REQ[READ.CSR] ADRS[CSR2] DT[LONG]
                ;22488                                           ;SET UP TO READ UB CSR
U 1D3A, 3022,15 ;22489          MOV MEM.DATA TO WR[0]            ;GET CONTENTS OF CSR 2 (UB ERROR REG)
U 1D3B, 2F82,95 ;22490          CLR WR[1]                        ;EXPECTED DATA
U 1D3C, 0869,3C ;22491          JSR [CHECK.RESULT]               ;CHECK FOR ALL ERROR BITS CLEAR
U 1D3D, 89D1,D4 ;22492          JMP [LOOP.T38.4]                 ;ERROR LOOP IF ENABLED
U 1D3E, 99BC,35 ;22493          MEM.REQ[WRITE.P] ADRS[CLEAR.ERR.ADDR] DT[WORD]
                ;22494                                           ;SET UP TO CLEAR UBE ERROR BITS IF SET
U 1D3F, B29C,15 ;22495          WRITE.MEM LS[ZERO]               ;CLEAR UBE ERROR REG
U 1D40, 5B00,14 ;22496          RETURN                           ;DONE WITH NPR TEST FOR THIS DATA
                ;22497
                ;22498
                ;22499  T38.ERROR.1:
U 1D41, B66E,15 ;22500          MOV LS[NA.EXP.REC] TO WR[0]      ;SET BIT TO PRINT N/A UNDER EXP AND REC
U 1D42, 6D80,15 ;22501          BIS WR[0] TO LS[CONTROL.STATUS] ;WRITE IN CONTROL AND STATUS WORD. KEEP
                ;22502                                           ; 'OTHER' BIT SET TO PRINT UNDER OTHER
U 1D43, 2F82,95 ;22503          CLR WR[1]                        ;FAKE 0'S EXPECTED DATA
U 1D44, B69E,15 ;22504          MOV LS[ONES] TO WR[0]            ;FAKE 1'S RECEIVED DATA
U 1D45, 0869,3C ;22505          JSR [CHECK.RESULT]               ;CALL ERROR ROUTINE
U 1D46, 09D1,C4 ;22506          JMP [LOOP.T38.1]                 ;ERROR LOOP IF ENABLED
U 1D47, B670,15 ;22507          MOV LS[OTHER.DATA] TO WR[0]      ;SET UP TO PRINT UNDER OTHER DATA
```

```
U 1D48, 3E80,15 ;22508          MOV WR[0] TO LS[CONTROL.STATUS] ;WRITE IN CONTROL AND STATUS WORD
U 1D49, 09D2,C4 ;22509          JMP [T38.2]                     ;CONTINUE WITH TEST
                ;22510
                ;22511
                ;22512  END.T38:
```

```
;22513  .page   ''  TEST 39 UNIBUS DATOB And TWO MEM CYC Write to Array Test (M8391 MCT Module)''
;22514  ;++
;22515  ;
;22516  ; Test Description:
;22517  ;
;22518  ;       This test checks the logic and micro-code associated with DATOB writes
;22519  ;       from the UNIBUS to the array and DATO writes to the array which require
;22520  ;       two memory cycles.
;22521  ;         The first test checks a DATOB to each of the 4 bytes in a longword
;22522  ;       with BYTE OFFSET clear.  This tests the logic associated with the DATA
;22523  ;       ROTATOR CONROL PAL and the MDR AND DIR CONTROL PAL.  The micro-flow is
;22524  ;       identical to the the previous UBE Write To Array Test.
;22525  ;         The second part of the tests checks the logic and micro-code path
;22526  ;       two memory cycles are required.  This occurs when a DATO to address 2
;22527  ;       with BYTE OFFSET set, or when a DATOB to address 3 with BYTE OFFSET set
;22528  ;       is executed.  The data path used in this test is the same as in the
;22529  ;       previous test up to UB CYC WR ARY C8 where it takes the 2 MEM CYCLE
;22530  ;       path to UB CYC WR ARY C9.  This cycle writes the first longword in
;22531  ;       the array.  The following cycle continues that write and enables the UB
;22532  ;       latch.  The next cycle completes the write, sets 2ND MEM CYC, and inc-
;22533  ;       rements the VAR.  The ECC latches are opened in the next cycle and the
;22534  ;       MDR is enabled onto the array bus.  3 cycles later the next array loc-
;22535  ;       ation is read and the ERR SUM CLK is asserted.  This cycle branches on
;22536  ;       IC0.
;22537  ;         IC0 was cleared by SET DATI in the cycle before UB CYC WR ARY C7.  So
;22538  ;       the IC0 not path is taken to UB CYC WR ARY C16.  The next 2 cycles get
;22539  ;       the array data into the ECC latches.  In UB CYC WR ARY C18 a branch on
;22540  ;       ERR should take the ERR not path to UB CYC WR ARY C19.  The following
;22541  ;       cycles complete the write to array and exit as did the previous test
;22542  ;       from UB CYC WR ARY C21.
;22543  ;
;22544  ; Assumptions:
;22545  ;
;22546  ;       It is assumed that all other micro-diagnostics have run successfully.
;22547  ;       It is also assumed that the UBE present is known to be good.
;22548  ;
;22549  ; Test Steps:
;22550  ;
;22551  ;       1)  Set up error mask, error number, and module number in LS (for
;22552  ;           error printouts) and clear previous error number in LS.  Also
;22553  ;           issue DCLO to UBE to clear error bits.
;22554  ;       2)  Set up UNIBUS MAP to map virtual address 0 to physical address 0
;22555  ;           and set the VALID bit.
;22556  ;       3)  Clear the IPL bits in the PSW (allow all interrupts).  Write all
;22557  ;           0's at physical memory address 0 and 4.
;22558  ;       4)  Set up UBE data reg with all 1's, cycle count to 1 cycle (FFFF),
;22559  ;           and address reg to address 0.
;22560  ;       5)  Set up control reg 1 of the UBE to execute one DATOB cycle via an
;22561  ;           NPR and interrupt via a BR6 when done.  Also set the GO bit to
;22562  ;           start the UBE.
;22563  ;       6)  Loop waiting for BR interrupt from UBE signifying that UBE is done.
;22564  ;       7)  Check the identifier code when BR interrupt occurs.  If a BR 7 was
;22565  ;           received (indicates a UBE error), read the UBE error register and
;22566  ;           report the error.
;22567  ;       8)  Read memory address 0 and check for one byte of ones.
```

```
;22568  ;       9)  Read memory address 4 and check for all zeros.
;22569  ;       10) Write address 0 and 4 with longword of zeros and repeat steps 4 -
;22570  ;           9 with a new address until each byte has been written seperately.
;22571  ;       11) Set BYTE OFFSET bit in UBS MAP.
;22572  ;       12) Write address 0 and 4 with longword of zeros and repeat steps 4
;22573  ;           through 7 with a 3 in the UBE address reg.  Check for zeros at
;22574  ;           memory address 0 and 1 byte of ones in the low byte of memory
;22575  ;           address 4.
;22576  ;       13) Set up control reg 1 to execute a DATO instead of DATOB.
;22577  ;       14) Write address 0 and 4 with longword of zeros and repeat steps 5
;22578  ;           through 7 with a 2 in the UBE address reg.  Check for a byte of
;22579  ;           ones in the high byte of memory address 0 and 1 byte of ones in
;22580  ;           the low byte of memory address 4.
;22581  ;
;22582  ; Errors:
;22583  ;
;22584  ;       Note:  For errors 1 through 3, data under OTHER in error report is phy-
;22585  ;       sical byte address that the UBE is trying to write to in main memory.
;22586  ;       The error number indicates which longword is being checked.
;22587  ;
;22588  ;       Error 1 - BR7 interrupt occurred when doing DATOB (indicates UBE error).
;22589  ;                 Received data is contents of UBE control reg 2 (error reg-
;22590  ;                 ister).
;22591  ;       Error 2 - Main memory longword 0 failed to get written correctly with
;22592  ;                 one byte on DATOB from UBE.
;22593  ;       Error 3 - Main memory longword at address 4 was modified by DOTOB to
;22594  ;                 byte in longword 0.
;22595  ;       Error 4 - BR7 interrupt occurred when doing DATOB to byte 3 with BYTE
;22596  ;                 OFFSET set (indicates UBE error).  Received data is contents
;22597  ;                 of UBE control reg 2 (error register).
;22598  ;       Error 5 - Main memory longword 0 was modified by DATOB to byte 3 with
;22599  ;                 BYTE OFFSET set.
;22600  ;       Error 6 - Main memory byte 0 at address 4 failed to get written cor-
;22601  ;                 rectly by DATOB to address 3 with BYTE OFFSET set.
;22602  ;       Error 7 - BR7 interrupt occurred when doing DATO to byte 2 with BYTE
;22603  ;                 OFFSET set (indicates UBE error).  Received data is contents
;22604  ;                 of UBE control reg 2 (error register).
;22605  ;       Error 8 - Main memory longword 0 byte 3 was not written correctly on
;22606  ;                 DATO to address 2 with BYTE OFFSET set.
;22607  ;       Error 9 - Main memory byte 0 at address 4 failed to get written cor-
;22608  ;                 rectly by DATO to address 2 with BYTE OFFSET set.
;22609  ;
;22610  ; Debug:
;22611  ;
;22612  ;       Error 1 - This error indicates that the UBE detected an error.  See the
;22613  ;       UBE error register list in this listing to determine what went wrong.
;22614  ;
;22615  ;       Error 2 - This error indicates that the UBE failed to write a byte of
;22616  ;       data correctly in the array.  The most likely suspect is the DATA
;22617  ;       ROTATOR PAL or the MDR AND DIR CONTROL PAL.
;22618  ;
;22619  ;       OTHER=0
;22620  ;
;22621  ;         If the result is shifted or more than one byte is written, suspect
;22622  ;       the DATA ROTATOR CONTROL PAL or the MDR AND DIR CONTROL PAL. The inputs
```

```
;22623  ;       to the MDR AND DIR CONTROL PAL during the WRITE SUBSTITUTE cycles
;22624  ;       should be as follows: RS1 L and RS0 L should both be high, A0 L and A1
;22625  ;       L should both be high, ROT C0 H should be high, CPH/UBL should be low,
;22626  ;       and 2ND MEM CYC H should be low.  The output should be low to the input
;22627  ;       of LAT OUTPUT BYT 1 L, LAT OUTPUT BYT 2 L, and LAT OUTPUT BYT 3 L. The
;22628  ;       output should be high to the input of LAT OUTPUT BYT 0 L.  If the in-
;22629  ;       puts are ok but the output are wrong, the PAL is probably bad.
;22630  ;         If the result is shifted or only one byte is written, and the inputs
;22631  ;       to the MDR AND DIR CONTROL PAL from the DATA ROTATOR PAL are not as
;22632  ;       descibed above, check the DATA ROTATOR CONTROL PAL for the following
;22633  ;       inputs during the cycle where ROT CLOCK is asserted.  CPH/UBL should be
;22634  ;       low, BYTE OFFSET H should be low, LVA 01 H should be low, and LUBC0 H
;22635  ;       should be high.  If these inputs are OK, the PAL is probably bad.
;22636  ;
;22637  ;       OTHER=1
;22638  ;
;22639  ;         The difference between writing at byte 1 from writing at byte 0 is
;22640  ;       that LVA 00 H going into the MDR AND DIR CONTROL PAL is high.  The
;22641  ;       other inputs are as above.  This changes the output so that the input
;22642  ;       to LAT OUTPUT BYT 1 L is high, while the inputs to the other LAT OUT-
;22643  ;       PUT BYTE gates are low.  The output and inputs to the DATA ROTATOR
;22644  ;       CONTROL PAL are as in OTHER=0 above.
;22645  ;
;22646  ;       OTHER=2
;22647  ;
;22648  ;         The difference between writing at byte 2 from writing at byte 0 is
;22649  ;       that A1 L is low and LVA 00 H must be low going into the MDR AND DIR
;22650  ;       CONTROL PAL.  The other inputs are as in OTHER=0 above.    This changes
;22651  ;       the output so that the input to LAT OUTPUT BYT 2 L is high, while the
;22652  ;       inputs to the other LAT OUTPUT BYTE gates are low.  The inputs to the
;22653  ;       DATA ROTATOR PAL are the same as OTHER=0 above except that LVA 01 H is
;22654  ;       high.  This changes the output A1 L to a low.
;22655  ;
;22656  ;       OTHER=3
;22657  ;
;22658  ;         The difference between writing at byte 3 from writing at byte 0 is
;22659  ;       that A1 L is low and LVA 00 H must be high going into the MDR AND DIR
;22660  ;       CONTROL PAL.  The other inputs are as in OTHER=0 above.    This changes
;22661  ;       the output so that the input to LAT OUTPUT BYT 3 L is high, while the
;22662  ;       inputs to the other LAT OUTPUT BYTE gates are low.  The inputs to the
;22663  ;       DATA ROTATOR PAL are the same as OTHER=0 above except that LVA 01 H is
;22664  ;       high.  This changes the output A1 L to a low.
;22665  ;
;22666  ;       Error 3 - This error indicates a problem with the UNIBUS WRITE TO ARRAY
;22667  ;       MICROCODE or the DATA ROTATOR CONTROL PAL.  Check that 2 MEM CYCLES L
;22668  ;       is not being asserted on the output of the PAL.  If it is, check the
;22669  ;       input BYTE OFFSET H for a low during the cycle that asserts ROT CLOCK
;22670  ;       H.  If BYTE OFFSET H is low, the PAL is probably bad.
;22671  ;
;22672  ;       Error 4 - This error indicates that the UBE detected an error.  See the
;22673  ;       UBE error register list in this listing to determine what went wrong.
;22674  ;
;22675  ;       Error 5 - This error indicates a problem with the MDR AND DIR CONTROL
;22676  ;       PAL or the UNIBUS WRITE TO ARRAY micro-flow when the 2 MEM CYC path is
;22677  ;       taken.  The micro-code path is described in the test Description above.
```

```
                ;22678  ;       If a byte in longword 0 is getting written, the most likely suspect is
                ;22679  ;       the PAL.  Check that A1 L and A0 L are both low going into the MDR AND
                ;22680  ;       DIR CONTROL PAL during the cycle that sets WRITE SUBSTITUTE.  If they
                ;22681  ;       are, the MDR AND DIR CONTROL PAL is probably bad.
                ;22682  ;
                ;22683  ;       Error 6 - This error indicates a problem with the DATA ROTATOR CONTROL
                ;22684  ;       PAL, the MDR AND DIR CONTROL PAL, or the UNIBUS WRITE TO ARRAY micro-
                ;22685  ;       flow when the 2 MEM CYC path is taken.  Check the signal 2 MEM CYCLES
                ;22686  ;       L out of the DATA ROTATOR CONTROL PAL for a low after the cycle that
                ;22687  ;       asserts ROT CLOCK H.  If incorrect, the PAL is probably bad.
                ;22688  ;         If 2 MEM CYCLES L is ok, check the input 2ND MEM CYC H to the MDR AND
                ;22689  ;       DIR CONTROL PAL during the cycle UB CYC WR ARY C18.  If it is high, the
                ;22690  ;       MDR AND DIR CONTROL PAL is probably bad.
                ;22691  ;
                ;22692  ;       Error 7 - This error indicates that the UBE detected an error.  See the
                ;22693  ;       UBE error register list in this listing to determine what went wrong.
                ;22694  ;
                ;22695  ;       Error 8 - Same as error 5.  Longword 0 should have the third byte writ-
                ;22696  ;       ten.
                ;22697  ;
                ;22698  ;       Error 9 - Same as error 6.
                ;22699  ;
                ;22700  ;--
                ;22701
                ;22702  ;       T14 = DATA FOR UBE CONTROL REG 1
                ;22703  ;       T15 = EXPECTED ARRAY DATA AT ADDRESS 0
                ;22704  ;       T57 = EXPECTED ARRAY DATA AT ADDRESS 4
                ;22705  ;       T58 = ADDRESS TO WRITE IN UBE ADDRESS REG
                ;22706
                ;22707  T.39:
U 1D4A, B65E,15 ;22708          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;22709                                          ; STATUS WORD
U 1D4B, 3E80,15 ;22710          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;22711                                          ; BIT 15 INDICATES START OF TEST TO
                ;22712                                          ; CONSOLE
U 1D4C, 10E0,15 ;22713          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;22714  WAIT.T39.0:
U 1D4D, 89D4,D4 ;22715          JMP [WAIT.T39.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;22716                                          ; RESPOND
                ;22717
U 1D4E, 0A1B,8C ;22718          JSR [ISSUE.DCLO]                ;INIT UBE
U 1D4F, 65A0,15 ;22719          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 1D50, 3644,15 ;22720          MOV LS[MCT] TO WR[0]            ;SET BIT 2 IN WR
U 1D51, 3E8C,15 ;22721          MOV WR[0] TO LS[MODULE.NUM]     ;SET UP TO CALL OUT MCT MODULE IF ERROR
U 1D52, 1B9D,F5 ;22722          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;22723                                          ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1D53, B27E,15 ;22724          WRITE.MEM LS[BIT31]             ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;22725                                          ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                ;22726                                          ; OFFSET CLEAR
U 1D54, 2F80,15 ;22727          CLR WR[0]                       ;CLEAR WR 0
U 1D55, BFFE,15 ;22728          MOV WR[0] TO LS[PSL.HW]         ;CLEAR IPL BITS IN PSW
U 1D56, 999C,75 ;22729          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;22730                                          ;SET UP TO WRITE AT MEMORY ADDR 0
U 1D57, B29C,15 ;22731          WRITE.MEM LS[ZERO]              ;WRITE ALL 0'S AT ADDR 0
U 1D58, 9944,75 ;22732          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
```

```
                ;22733                                          ;SET UP TO WRITE AT MEMORY ADDR 4
U 1D59, B29C,15 ;22734          WRITE.MEM LS[ZERO]              ;WRITE ALL 0'S AT ADDR 4
                ;22735
U 1D5A, 0A1C,0C ;22736          JSR [SETUP.DATO.NPR]            ;SETUP T14 WITH UBE DATA TO ISSUE NPR
                ;22737                                          ; DATO, INTERRUPT ON DONE AT BR6
U 1D5B, 3650,15 ;22738          MOV LS[C00] TO WR[0]            ;SET BIT 8 IN WR (C00 LINE)
U 1D5C, 6D1C,15 ;22739          BIS WR[0] TO LS[T14]            ;SET C00 TO MAKE DATO INTO DATOB IN UBE
                ;22740
U 1D5D, B670,15 ;22741          MOV LS[OTHER.DATA] TO WR[0]     ;SET BIT TO PRINT UNDER OTHER DATA
U 1D5E, 3E80,15 ;22742          MOV WR[0] TO LS[CONTROL.STATUS] ;WRITE IN CONTROL AND STATUS WORD
                ;22743
U 1D5F, 3641,15 ;22744          MOV LS[#1] TO WR[2]             ;STARTING ERROR NUMBER (ERROR 1)
U 1D60, 1984,35 ;22745          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                ;22746                                          ;SET UP TO WRITE TO UBE DATA REG
U 1D61, 329E,15 ;22747          WRITE.MEM LS[ONES]              ;PUT 1'S IN DATA REG
U 1D62, 6588,15 ;22748          CLR LS[ADDRESS.DATA]            ;0 IN OTHER DATA (BYTE BEING WRITTEN)
U 1D63, E580,15 ;22749          CLR LS[T58]                     ;CLEAR DATA TO WRITE IN UBE ADDRESS REG
U 1D64, B626,15 ;22750          MOV LS[#FF] TO WR[0]            ;FF IN WR
U 1D65, BE1E,15 ;22751          MOV WR[0] TO LS[T15]            ;EXPECTED DATA
U 1D66, E5AE,15 ;22752          CLR LS[T57]                     ;EXPECTED DATA FOR 2ND LONGWORD
U 1D67, 89DA,1C ;22753          JSR [LOOP.T39.2.3]              ;PERFORM TEST WITH ALL 1'S DATA
                ;22754                                          ; TO ADDRESS 0 (BYTE 0)
                ;22755
U 1D68, 999C,75 ;22756          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;22757                                          ;SET UP TO WRITE ARRAY ADDRESS 0
U 1D69, B29C,15 ;22758          WRITE.MEM LS[ZERO]              ;WRITE LONGWORD OF 0'S AT ADDRESS 0
U 1D6A, 9944,75 ;22759          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                ;22760                                          ;SET UP TO WRITE AT MEMORY ADDR 4
U 1D6B, B29C,15 ;22761          WRITE.MEM LS[ZERO]              ;WRITE ALL 0'S AT ADDR 0
U 1D6C, FF88,15 ;22762          INC LS[ADDRESS.DATA]            ;1 IN OTHER DATA (BYTE BEING WRITTEN)
U 1D6D, 7FB0,15 ;22763          INC LS[T58]                     ;DATA OF 1 TO WRITE IN UBE ADDRESS REG
U 1D6E, 3628,15 ;22764          MOV LS[#FFFF] TO WR[0]          ;0000FFFF IN WR
U 1D6F, 4526,15 ;22765          BIC LS[#FF] TO WR[0]            ;0000FF00 IN WR
U 1D70, BE1E,15 ;22766          MOV WR[0] TO LS[T15]            ;EXPECTED DATA
U 1D71, 89DA,1C ;22767          JSR [LOOP.T39.2.3]              ;PERFORM TEST WITH ALL 1'S DATA
                ;22768                                          ; TO ADDRESS 1 (BYTE 1)
                ;22769
U 1D72, 999C,75 ;22770          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;22771                                          ;SET UP TO WRITE ARRAY ADDRESS 0
U 1D73, B29C,15 ;22772          WRITE.MEM LS[ZERO]              ;WRITE LONGWORD OF 0'S AT ADDRESS 0
U 1D74, 9944,75 ;22773          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                ;22774                                          ;SET UP TO WRITE AT MEMORY ADDR 4
U 1D75, B29C,15 ;22775          WRITE.MEM LS[ZERO]              ;WRITE ALL 0'S AT ADDR 0
U 1D76, FF88,15 ;22776          INC LS[ADDRESS.DATA]            ;2 IN OTHER DATA (BYTE BEING WRITTEN)
U 1D77, 7FB0,15 ;22777          INC LS[T58]                     ,DATA OF 2 TO WRITE IN UBE ADDRESS REG
U 1D78, DF2A,15 ;22778          MCOM LS[#FF000000] TO WR[0]     ;00FFFFFF IN WR
U 1D79, C528,15 ;22779          BIC LS[#FFFF] TO WR[0]          ;00FF0000 IN WR
U 1D7A, BE1E,15 ;22780          MOV WR[0] TO LS[T15]            ;EXPECTED DATA
U 1D7B, 89DA,1C ;22781          JSR [LOOP.T39.2.3]              ;PERFORM TEST WITH ALL 1'S DATA
                ;22782                                          ; TO ADDRESS 2 (BYTE 2)
                ;22783
U 1D7C, 999C,75 ;22784          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;22785                                          ;SET UP TO WRITE ARRAY ADDRESS 0
U 1D7D, B29C,15 ;22786          WRITE.MEM LS[ZERO]              ;WRITE LONGWORD OF 0'S AT ADDRESS 0
U 1D7E, 9944,75 ;22787          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
```

```
                ;22788                                  ;SET UP TO WRITE AT MEMORY ADDR 4
U 1D7F, B29C,15 ;22789          WRITE.MEM LS[ZERO]      ;WRITE ALL 0'S AT ADDR 0
U 1D80, FF88,15 ;22790          INC LS[ADDRESS.DATA]    ;3 IN OTHER DATA (BYTE BEING WRITTEN)
U 1D81, 7FB0,15 ;22791          INC LS[T58]             ;DATA OF 3 TO WRITE IN UBE ADDRESS REG
U 1D82, B62A,15 ;22792          MOV LS[#FF000000] TO WR[0] ;FF000000 IN WR
U 1D83, BE1E,15 ;22793          MOV WR[0] TO LS[T15]    ;EXPECTED DATA
U 1D84, 89DA,1C ;22794          JSR [LOOP.T39.2.3]      ;PERFORM TEST WITH ALL 1'S DATA
                ;22795                                  ; TO ADDRESS 3 (BYTE 3)
                ;22796
U 1D85, 367E,15 ;22797          MOV LS[BIT31] TO WR[0]  ;SET BIT 31 IN WR (VALID BIT)
U 1D86, 4772,15 ;22798          BIS LS[BIT25] TO WR[0]  ;SET BIT 25 IN WR (BYTE OFFSET)
U 1D87, BEB2,15 ;22799          MOV WR[0] TO LS[T59]    ;STORE IN LS
U 1D88, 1B9D,F5 ;22800          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;22801                                  ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1D89, B2B2,15 ;22802          WRITE.MEM LS[T59]       ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;22803                                  ; ADDR 0 TO PHYSICAL ADDR 0, SET BYTE
                ;22804                                  ; OFFSET BIT
                ;22805
U 1D8A, E580,15 ;22806          CLR LS[CONTROL.STATUS]  ;DISABLE OTHER PRINTOUT
U 1D8B, B645,15 ;22807          MOV LS[#4] TO WR[2]     ;STARTING ERROR NUMBER (ERROR 4)
U 1D8C, 999C,75 ;22808          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;22809                                  ;SET UP TO WRITE ARRAY ADDRESS 0
U 1D8D, B29C,15 ;22810          WRITE.MEM LS[ZERO]      ;WRITE LONGWORD OF 0'S AT ADDRESS 0
U 1D8E, 9944,75 ;22811          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                ;22812                                  ;SET UP TO WRITE ARRAY ADDRESS 4
U 1D8F, B29C,15 ;22813          WRITE.MEM LS[ZERO]      ;WRITE LONGWORD OF 0'S AT ADDRESS 4
U 1D90, 651E,15 ;22814          CLR LS[T15]             ;EXPECTED DATA AT ADDRESS 0
U 1D91, B626,15 ;22815          MOV LS[#FF] TO WR[0]    ;000000FF IN WR
U 1D92, 3EAE,15 ;22816          MOV WR[0] TO LS[T57]    ;EXPECTED DATA AT ADDRESS 4
U 1D93, 89DA,1C ;22817          JSR [LOOP.T39.2.3]      ;PERFORM TEST WITH ALL 1'S DATA
                ;22818                                  ; TO ADDRESS 4 (DATOB TO 3 WITH BYTE
                ;22819                                  ; OFFSET SET)
                ;22820
U 1D94, 0A1C,0C ;22821          JSR [SETUP.DATO.NPR]    ;SETUP T14 WITH UBE DATA TO ISSUE NPR
                ;22822                                  ; DATO, INTERRUPT ON DONE AT BR6
U 1D95, 40C1,15 ;22823          ADD LS[#3(H)] TO WR[2]  ;STARTING ERROR NUMBER (ERROR 7)
U 1D96, 999C,75 ;22824          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;22825                                  ;SET UP TO WRITE ARRAY ADDRESS 0
U 1D97, B29C,15 ;22826          WRITE.MEM LS[ZERO]      ;WRITE LONGWORD OF 0'S AT ADDRESS 0
U 1D98, 9944,75 ;22827          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                ;22828                                  ;SET UP TO WRITE ARRAY ADDRESS 4
U 1D99, B29C,15 ;22829          WRITE.MEM LS[ZERO]      ;WRITE LONGWORD OF 0'S AT ADDRESS 4
U 1D9A, 7CB0,15 ;22830          DEC LS[T58]             ;DATA OF 2 TO WRITE IN ADDRESS REG
U 1D9B, B62A,15 ;22831          MOV LS[#FF000000] TO WR[0] ;FF000000 IN WR
U 1D9C, BE1E,15 ;22832          MOV WR[0] TO LS[T15]    ;EXPECTED DATA AT ADDRESS 0
U 1D9D, B626,15 ;22833          MOV LS[#FF] TO WR[0]    ;000000FF IN WR
U 1D9E, 3EAE,15 ;22834          MOV WR[0] TO LS[T57]    ;EXPECTED DATA AT ADDRESS 4
U 1D9F, 89DA,1C ;22835          JSR [LOOP.T39.2.3]      ;PERFORM TEST WITH ALL 1'S DATA
                ;22836                                  ; TO ADDRESS 4 (DATO TO 2 WITH BYTE
                ;22837                                  ; OFFSET SET)
                ;22838
U 1DA0, 89DB,B4 ;22839          JMP [END.T39]           ;DONE
                ;22840
                ;22841  ;
                ;22842  ;SUBROUTINE TO PERFORM NPR TEST
```

```
                ;22843  ;
                ;22844
                ;22845  LOOP.T39.2.3:
U 1DA1, 3E83,15 ;22846          MOV WR[2] TO LS[ERROR.NUMBER]   ;SET UP ERR NUMBER IN LS (ERROR 1,4 OR
                ;22847                                          ; 7)
                ;22848  LOOP.T39.1:
U 1DA2, 19B8,35 ;22849          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                ;22850                                          ;SET UP TO WRITE TO UBE ADDRESS REG
U 1DA3, 32B0,15 ;22851          WRITE.MEM LS[T58]               ;PUT CURRENT ADDRESS IN ADDRESS REG
U 1DA4, 99B6,35 ;22852          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                ;22853                                          ;SET UP TO WRITE TO UBE CYCLE COUNT
U 1DA5, 329E,15 ;22854          WRITE.MEM LS[ONES]              ;PUT A -1 IN CYCLE COUNT (DO 1 CYCLE)
U 1DA6, 99BA,35 ;22855          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                ;22856                                          ;SET UP TO WRITE TO UBE CONTROL REG 1
U 1DA7, 321C,15 ;22857          WRITE.MEM LS[T14]               ;SET UP CONTROL REG TO ISSUE NPR TO
                ;22858                                          ; WRITE A BYTE TO ARRAY AND INTR AT
                ;22859                                          ; BR6 ON DONE
                ;22860  LOOP.T39.INT.1:
U 1DA8, 09DA,87 ;22861          JMP.IF[NO.INTERRUPT] TO [LOOP.T39.INT.1] ;LOOP UNTIL BR INTERRUPT
                ;22862
U 1DA9, 8A1C,DC ;22863          JSR [CHECK.UBE.ERR]             ;SEE IF UBE ERR AND ISSUE NECESSARY
                ;22864                                          ; GRANTS
U 1DAA, 09DA,24 ;22865          JMP [LOOP.T39.1]                ;ERROR LOOP RETURN IF ENABLED
                ;22866
U 1DAB, FF82,15 ;22867          INC LS[ERROR.NUMBER]            ;ERROR 2, 5, OR 8
U 1DAC, E58A,15 ;22868          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
U 1DAD, 189D,F5 ;22869          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;22870                                          ;SET UP TO READ DATA AT ADDRESS 0
U 1DAE, 3022,15 ;22871          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1DAF, B61E,95 ;22872          MOV LS[T15] TO WR[1]            ;EXPECTED DATA
U 1DB0, 0869,3C ;22873          JSR [CHECK.RESULT]              ;SEE IF ARRAY WRITTEN CORRECTLY
U 1DB1, 09DA,14 ;22874          JMP [LOOP.T39.2.3]              ;ERROR LOOP IF ENABLED
                ;22875
U 1DB2, FF82,15 ;22876          INC LS[ERROR.NUMBER]            ;ERROR 3, 6, OR 9
U 1DB3, 1845,F5 ;22877          MEM.REQ[READ.P] ADRS[#4] DT[LONG]
                ;22878                                          ;SET UP TO READ DATA AT ADDRESS 4
U 1DB4, 3022,15 ;22879          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1DB5, 36AE,95 ;22880          MOV LS[T57] TO WR[1]            ;EXPECTED DATA
U 1DB6, 0869,3C ;22881          JSR [CHECK.RESULT]              ;SEE IF ARRAY WRITTEN CORRECTLY
U 1DB7, 09DA,14 ;22882          JMP [LOOP.T39.2.3]              ;ERROR LOOP IF ENABLED
                ;22883
U 1DB8, 99BC,35 ;22884          MEM.REQ[WRITE.P] ADRS[CLEAR.ERR.ADDR] DT[WORD]
                ;22885                                          ;SET UP TO CLEAR UBE ERROR BITS IF SET
U 1DB9, B29C,15 ;22886          WRITE.MEM LS[ZERO]              ;CLEAR UBE ERROR REG
U 1DBA, 5B00,14 ;22887          RETURN                          ;DONE WITH NPR TEST FOR THIS DATA
                ;22888
                ;22889  END.T39:
```

```
;22890  .page   "  TEST 3A UNIBUS DATO With CRD and RDS Error Test (M8391 MCT Module)"
;22891  ;++
;22892  ;
;22893  ; Test Description:
;22894  ;
;22895  ;       This test checks the micro-code associated with DATO writes from the
;22896  ;       UNIBUS to the array when single or double bit errors are detected.
;22897  ;         The test writes single bit errors in memory and attempts to write
;22898  ;       to that location.  The micro-code should branch to the routine to
;22899  ;       correct the single bit error before writing the new data back into
;22900  ;       memory.
;22901  ;         After this has been verified, double bits errors are written into
;22902  ;       memory and the flow which aborts the UNIBUS write to array is checked.
;22903  ;       When the double bit error occurs in the second half of a 2 mem cyc
;22904  ;       write, the first memory cycle is checked for normal completion.
;22905  ;
;22906  ; Assumptions:
;22907  ;
;22908  ;       It is assumed that all other micro-diagnostics have run successfully.
;22909  ;       It is also assumed that the UBE present is known to be good.
;22910  ;
;22911  ; Test Steps:
;22912  ;
;22913  ;       1)  Set up error mask, error number, and module number in LS (for
;22914  ;           error printouts) and clear previous error number in LS.  Also
;22915  ;           issue DCLO to UBE to clear error bits.
;22916  ;       2)  Set up UNIBUS MAP to map virtual address 0 to physical address 0
;22917  ;           and set the VALID and BYTE OFFSET bits.
;22918  ;       3)  Clear the IPL bits in the PSW (allow all interrupts).  Write a
;22919  ;           10000 at physical memory address 0 and 4 with CKBTS for data of
;22920  ;           all 0's (single bit error in byte 2 of each longword).
;22921  ;       4)  Set up UBE data reg with all 1's, cycle count to 1 cycle (FFFF),
;22922  ;           and address reg to address 2.
;22923  ;       5)  Set up control reg 1 of the UBE to execute one DATO cycle via an
;22924  ;           NPR and interrupt via a BR6 when done.  Also set the GO bit to
;22925  ;           start the UBE.
;22926  ;       6)  Loop waiting for BR interrupt from UBE signifying that UBE is done.
;22927  ;       7)  Check the identifier code when BR interrupt occurs.  If a BR 7 was
;22928  ;           received (indicates a UBE error), read the UBE error register and
;22929  ;           report the error.
;22930  ;       8)  Read memory address 0 and check for 1 byte of ones at byte 3.
;22931  ;       9)  Read memory address 4 and check for 1 byte of ones at byte 0.
;22932  ;       10) Write address 0 with data of 30000 and CKBTS for data of all zeros.
;22933  ;           Write address 4 with all 0's and normal CKBTS.  Repeat steps 4
;22934  ;           through 7 above.
;22935  ;       11) Read memory address 0 and check for data of 30000 (no change).
;22936  ;       12) Read memory address 4 and check for all 0's (no change).
;22937  ;       13) Write address 0 with data of all 0's and normal CKBTs.  Write ad-
;22938  ;           dress 4 with data of 30000 and CKBTs for data of all 0's.  Repeat
;22939  ;           steps 4 through 7 above.
;22940  ;       14) Read memory address 0 and check for 1 byte of ones at byte 3.
;22941  ;       15) Read memory address 4 and check for data of 30000 (no change).
;22942  ;
;22943  ; Errors:
;22944  ;
```

```
;22945  ;
;22946  ;       Error 1 - BR7 interrupt occurred when doing DATO (indicates UBE error).
;22947  ;                 Received data is contents of UBE control reg 2 (error reg-
;22948  ;                 ister).
;22949  ;       Error 2 - Main memory longword 0 failed to get single bit error cor-
;22950  ;                 rected and/or written with 1 byte on DATO from UBE.
;22951  ;       Error 3 - Main memory longword 4 failed to get single bit error cor-
;22952  ;                 rected and/or written with 1 byte on DATO from UBE.
;22953  ;       Error 4 - BR7 interrupt occurred when doing DATO (indicates UBE error).
;22954  ;                 Received data is contents of UBE control reg 2 (error reg-
;22955  ;                 ister).
;22956  ;       Error 5 - Main memory longword 0 was modified by DATO from UBE when
;22957  ;                 array contained RDS error in longword 0.
;22958  ;       Error 6 - Main memory longword 4 was modified by DATO from UBE when
;22959  ;                 array contained RDS error in longword 0.
;22960  ;       Error 7 - BR7 interrupt occurred when doing DATO (indicates UBE error).
;22961  ;                 Received data is contents of UBE control reg 2 (error reg-
;22962  ;                 ister).
;22963  ;       Error 8 - Main memory longword 0 was not written on DATO from UBE when
;22964  ;                 array contained RDS error in longword at address 4.
;22965  ;       Error 9 - Main memory longword 4 was modified by DATO from UBE when
;22966  ;                 array contained RDS error in longword at address 4.
;22967  ;
;22968  ; Debug:
;22969  ;
;22970  ;       Error 1 - This error indicates that the UBE detected an error.  See the
;22971  ;       UBE error register list in this listing to determine what went wrong.
;22972  ;
;22973  ;       Error 2 - This error most likely indicates a problem with the micro-
;22974  ;       code.  The micro-flow should take the ERR path at UB CYC WR ARY C7 to
;22975  ;       UB CYC WR ARY WDE C9.  From there the SERR path should be taken to UB
;22976  ;       CYC WR ARY WSDE C8A.  Next UB CYC WR ARY WSDE C9 should be executed,
;22977  ;       followed by UB CYC WR ARY C8.
;22978  ;
;22979  ;       Error 3 - This error most likely indicates a problem with the micro-
;22980  ;       code.  The micro-flow should take the ERR path at UB CYC WR ARY C18 to
;22981  ;       UB CYC WR ARY WDE C19.  From there the SERR path should be taken to UB
;22982  ;       CYC WR ARY WSDE C20.  Next UB CYC WR ARY WSDE C21 should be executed,
;22983  ;
;22984  ;       Error 4 - This error indicates that the UBE detected an error.  See the
;22985  ;       UBE error register list in this listing to determine what went wrong.
;22986  ;
;22987  ;       Error 5 - This error most likely indicates a problem with the micro-
;22988  ;       code.  The micro-flow should take the ERR path at UB CYC WR ARY C7 to
;22989  ;       UB CYC WR ARY WDE C9.  From there the SERR not path should be taken to
;22990  ;       UB CYC ABORT WR WUDE followed by UB CYC ABORT WR WUDE C7.  The cycle
;22991  ;       following this will then go to UB CYC WR ARY C22 thus aborting both
;22992  ;       memory cycles.
;22993  ;
;22994  ;       Error 6 - Same as error 5.  The second memory cycle should never have
;22995  ;       been executed.
;22996  ;
;22997  ;       Error 7 - This error indicates that the UBE detected an error.  See the
;22998  ;       UBE error register list in this listing to determine what went wrong.
;22999  ;
```

```
                ;23000  ;       Error 8 - This error should not occur as it takes the normal path for
                ;23001  ;       the first memory cycle.  Run previous tests again if this error occurs.
                ;23002  ;
                ;23003  ;       Error 9 - This error most likely indicates a problem with the micro-
                ;23004  ;       code.  The micro-flow should take the ERR path at UB CYC WR ARY C18 to
                ;23005  ;       UB CYC WR ARY WDE C19.  From there the SERR not path should be taken to
                ;23006  ;       the cycle that goes to UB CYC ABORT WR WUDE C7.  The cycle following
                ;23007  ;       this will then go to UB CYC WR ARY C22 thus aborting the second memory
                ;23008  ;       cycle.
                ;23009  ;
                ;23010  ;--
                ;23011
                ;23012  ;       T14 = DATA FOR UBE CONTROL REG 1
                ;23013  ;       T15 = EXPECTED ARRAY DATA AT ADDRESS 0
                ;23014  ;       T57 = EXPECTED ARRAY DATA AT ADDRESS 4
                ;23015  ;       T58 = ADDRESS 5 FOR WRITING ERRORS IN LONGWORD 5
                ;23016  ;       DATA.1 = BACKGROUND ARRAY DATA FOR ADDRESS 0
                ;23017  ;       DATA.2 = BACKGROUND ARRAY DATA FOR ADDRESS 4
                ;23018
                ;23019  T.3A:
U 1DBB, B65E,15 ;23020          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;23021                                          ; STATUS WORD
U 1DBC, 3E80,15 ;23022          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;23023                                          ; BIT 15 INDICATES START OF TEST TO
                ;23024                                          ; CONSOLE
U 1DBD, 10E0,15 ;23025          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;23026  WAIT.T3A.0:
U 1DBE, 89DB,E4 ;23027          JMP [WAIT.T3A.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;23028                                          ; RESPOND
                ;23029
U 1DBF, 0A1B,8C ;23030          JSR [ISSUE.DCLO]                ;INIT UBE
U 1DC0, 65A0,15 ;23031          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 1DC1, 3644,15 ;23032          MOV LS[MCT] TO WR[0]            ;SET BIT 2 IN WR
U 1DC2, 3E8C,15 ;23033          MOV WR[0] TO LS[MODULE.NUM]     ;SET UP TO CALL OUT MCT MODULE IF ERROR
U 1DC3, 367E,15 ;23034          MOV LS[BIT31] TO WR[0]          ;SET BIT 31 IN WR
U 1DC4, 4772,15 ;23035          BIS LS[BIT25] TO WR[0]          ;SET BIT 25 IN WR (BYTE OFFSET)
U 1DC5, 3E10,15 ;23036          MOV WR[0] TO LS[T8]             ;STORE IN LS
U 1DC6, 1B9D,F5 ;23037          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;23038                                          ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1DC7, 3210,15 ;23039          WRITE.MEM LS[T8]                ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;23040                                          ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                ;23041                                          ; OFFSET SET
U 1DC8, 2F80,15 ;23042          CLR WR[0]                       ;CLEAR WR 0
U 1DC9, BFFE,15 ;23043          MOV WR[0] TO LS[PSL.HW]         ;CLEAR IPL BITS IN PSW
U 1DCA, B700,15 ;23044          MOV LS[CKBTS.01111100] TO WR[0] ;GET CKBTS FOR DATA OF 0 IN WR 0
U 1DCB, A0C0,15 ;23045          COM WR[0]                       ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 1DCC, 452C,15 ;23046          BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE
U 1DCD, 8A17,FC ;23047          JSR [WRITE.CSR1]                ;PUT CKBTS FOR DATA OF 0 IN CSR 1, CLEAR
                ;23048                                          ; ALL OTHER BITS IN CSR
U 1DCE, 3644,15 ;23049          MOV LS[#4] TO WR[0]             ;4 IN WR
U 1DCF, 2040,15 ;23050          INC WR[0]                       ;5 IN WR
U 1DD0, 3EB0,15 ;23051          MOV WR[0] TO LS[T58]            ;5 IN LS
U 1DD1, 0A1C,0C ;23052          JSR [SETUP.DATO.NPR]            ;SETUP T14 WITH UBE DATA TO ISSUE NPR
                ;23053                                          ; DATO, INTERRUPT ON DONE AT BR6
                ;23054
```

```
U 1DD2, 3641,15 ;23055          MOV LS[#1] TO WR[2]              ;STARTING ERROR NUMBER (ERROR 1)
U 1DD3, 19B4,35 ;23056          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                ;23057                                           ;SET UP TO WRITE TO UBE DATA REG
U 1DD4, 329E,15 ;23058          WRITE.MEM LS[ONES]               ;PUT 1'S IN DATA REG
U 1DD5, 3660,15 ;23059          MOV LS[#10000] TO WR[0]          ;10000 IN WR
U 1DD6, 3EA2,15 ;23060          MOV WR[0] TO LS[DATA.1]          ;BACKGROUND DATA FOR ADDRESS 0
U 1DD7, 3EA4,15 ;23061          MOV WR[0] TO LS[DATA.2]          ;BACKGROUND DATA FOR ADDRESS 4
U 1DD8, B62A,15 ;23062          MOV LS[#FF000000] TO WR[0]       ;FF000000 IN WR
U 1DD9, BE1E,15 ;23063          MOV WR[0] TO LS[T15]             ;EXPECTED DATA IN FIRST LONGWORD
U 1DDA, B626,15 ;23064          MOV LS[#FF] TO WR[0]             ;000000FF IN WR
U 1DDB, 3EAE,15 ;23065          MOV WR[0] TO LS[T57]             ;EXPECTED DATA IN SECOND LONGWORD
U 1DDC, 09DE,EC ;23066          JSR [LOOP.T3A.2.3]               ;PERFORM TEST WITH ALL 1'S DATA
                ;23067                                           ; AND CRD ERRORS AT BOTH LOCATIONS
                ;23068
U 1DDD, B645,15 ;23069          MOV LS[#4] TO WR[2]              ;STARTING ERROR NUMBER (ERROR 4)
U 1DDE, 3660,15 ;23070          MOV LS[#10000] TO WR[0]          ;10000 IN WR
U 1DDF, C762,15 ;23071          BIS LS[#20000] TO WR[0]          ;30000 IN WR
U 1DE0, 3EA2,15 ;23072          MOV WR[0] TO LS[DATA.1]          ;BACKGROUND DATA FOR ADDRESS 0 (RDS
                ;23073                                           ; ERROR)
U 1DE1, E5A4,15 ;23074          CLR LS[DATA.2]                   ;BACKGROUND DATA FOR ADDRESS 4
U 1DE2, BE1E,15 ;23075          MOV WR[0] TO LS[T15]             ;EXPECTED DATA FOR ADDRESS 0 (30000(H))
U 1DE3, E5AE,15 ;23076          CLR LS[T57]                      ;EXPECTED DATA AT ADDRESS 4
U 1DE4, 09DE,EC ;23077          JSR [LOOP.T3A.2.3]               ;PERFORM TEST WITH ALL 1'S DATA
                ;23078                                           ; AND RDS ERROR AT ADDRESS 0
                ;23079
U 1DE5, 40C1,15 ;23080          ADD LS[#3(H)] TO WR[2]           ;STARTING ERROR NUMBER (ERROR 7)
U 1DE6, E5A2,15 ;23081          CLR LS[DATA.1]                   ;BACKGROUND DATA FOR ADDRESS 0
U 1DE7, 361E,15 ;23082          MOV LS[T15] TO WR[0]             ;30000(H) IN WR
U 1DE8, 3EA4,15 ;23083          MOV WR[0] TO LS[DATA.2]          ;BACKGROUND DATA FOR ADDRESS 4 (RDS
                ;23084                                           ; ERROR)
U 1DE9, 3EAE,15 ;23085          MOV WR[0] TO LS[T57]             ;EXPECTED DATA AT ADDRESS 4 (30000(H))
U 1DEA, B62A,15 ;23086          MOV LS[#FF000000] TO WR[0]       ;FF000000 IN WR
U 1DEB, BE1E,15 ;23087          MOV WR[0] TO LS[T15]             ;EXPECTED DATA FOR ADDRESS 0
U 1DEC, 09DE,EC ;23088          JSR [LOOP.T3A.2.3]               ;PERFORM TEST WITH ALL 1'S DATA
                ;23089                                           ; AND RDS ERROR AT ADDRESS 4
                ;23090
U 1DED, 89E0,C4 ;23091          JMP [END.T3A]                    ;DONE
                ;23092
                ;23093  ;
                ;23094  ;SUBROUTINE TO PERFORM NPR TEST
                ;23095  ;
                ;23096
                ;23097  LOOP.T3A.2.3:
U 1DEE, 3E83,15 ;23098          MOV WR[2] TO LS[ERROR.NUMBER]    ;SET UP ERR NUMBER IN LS (ERROR 1,4 OR
                ;23099                                           ; 7)
                ;23100  LOOP.T3A.1:
U 1DEF, 1D40,F5 ;23101          MEM.REQ[MAINT.ECC.DATA] ADRS[#1] DT[LONG]
                ;23102                                           ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;23103                                           ; SET FOR BRANCH
U 1DF0, 32A2,15 ;23104          WRITE.MEM LS[DATA.1]             ;WRITE DATA AT LOCATION 0 WITH CKBTS
                ;23105                                           ; FOR DATA OF ALL 0'S
U 1DF1, 1DB0,F5 ;23106          MEM.REQ[MAINT.ECC.DATA] ADRS[T58] DT[LONG]
                ;23107                                           ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;23108                                           ; SET FOR BRANCH
U 1DF2, 32A4,15 ;23109          WRITE.MEM LS[DATA.2]             ;WRITE DATA AT LOCATION 4 WITH CKBTS
```

```
                    ;23110                                          ; FOR DATA OF ALL 0'S
U 1DF3, 19B8,35     ;23111          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                    ;23112                                          ;SET UP TO WRITE TO UBE ADDRESS REG
U 1DF4, B242,15     ;23113          WRITE.MEM LS[#2]                ;PUT ADDRESS 2 IN UBE ADDRESS REG
U 1DF5, 99B6,35     ;23114          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                    ;23115                                          ;SET UP TO WRITE TO UBE CYCLE COUNT
U 1DF6, 329E,15     ;23116          WRITE.MEM LS[ONES]              ;PUT A -1 IN CYCLE COUNT (DO 1 CYCLE)
U 1DF7, 99BA,35     ;23117          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                    ;23118                                          ;SET UP TO WRITE TO UBE CONTROL REG 1
U 1DF8, 321C,15     ;23119          WRITE.MEM LS[T14]               ;SET UP CONTROL REG TO ISSUE NPR TO
                    ;23120                                          ; WRITE A WORD TO ARRAY AND INTR AT
                    ;23121                                          ; BR6 ON DONE
                    ;23122  LOOP.T3A.INT.1:
U 1DF9, 89DF,97     ;23123          JMP.IF[NO.INTERRUPT] TO [LOOP.T3A.INT.1] ;LOOP UNTIL BR INTERRUPT
                    ;23124
U 1DFA, 8A1C,DC     ;23125          JSR [CHECK.UBE.ERR]             ;SEE IF UBE ERR AND ISSUE NECESSARY
                    ;23126                                          ; GRANTS
U 1DFB, 09DE,F4     ;23127          JMP [LOOP.T3A.1]                ;ERROR LOOP RETURN IF ENABLED
                    ;23128
U 1DFC, FF82,15     ;23129          INC LS[ERROR.NUMBER]            ;ERROR 2, 5, OR 8
U 1DFD, E58A,15     ;23130          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
U 1DFE, 189D,F5     ;23131          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                    ;23132                                          ;SET UP TO READ DATA AT ADDRESS 0
U 1DFF, 3022,15     ;23133          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1E00, B61E,95     ;23134          MOV LS[T15] TO WR[1]            ;EXPECTED DATA
U 1E01, 0869,3C     ;23135          JSR [CHECK.RESULT]              ;SEE IF ARRAY WRITTEN CORRECTLY
U 1E02, 89DE,E4     ;23136          JMP [LOOP.T3A.2.3]              ;ERROR LOOP IF ENABLED
                    ;23137
U 1E03, FF82,15     ;23138          INC LS[ERROR.NUMBER]            ;ERROR 3, 6, OR 9
U 1E04, 1845,F5     ;23139          MEM.REQ[READ.P] ADRS[#4] DT[LONG]
                    ;23140                                          ;SET UP TO READ DATA AT ADDRESS 4
U 1E05, 3022,15     ;23141          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1E06, 36AE,95     ;23142          MOV LS[T57] TO WR[1]            ;EXPECTED DATA
U 1E07, 0869,3C     ;23143          JSR [CHECK.RESULT]              ;SEE IF ARRAY WRITTEN CORRECTLY
U 1E08, 89DE,E4     ;23144          JMP [LOOP.T3A.2.3]              ;ERROR LOOP IF ENABLED
                    ;23145
U 1E09, 99BC,35     ;23146          MEM.REQ[WRITE.P] ADRS[CLEAR.ERR.ADDR] DT[WORD]
                    ;23147                                          ;SET UP TO CLEAR UBE ERROR BITS IF SET
U 1E0A, B29C,15     ;23148          WRITE.MEM LS[ZERO]              ;CLEAR UBE ERROR REG
U 1E0B, 5B00,14     ;23149          RETURN                          ;DONE WITH NPR TEST FOR THIS DATA
                    ;23150
                    ;23151  END.T3A:
```

```
;23152  .page   "  TEST 3B UNIBUS DATO Crossing Page Boundary Test (M8391 MCT Module)"
;23153  ;++
;23154  ;
;23155  ; Test Description:
;23156  ;
;23157  ;       This test checks the micro-code associated with DATO writes from the
;23158  ;       UNIBUS to the array when a page boundary is crossed.  The test is run
;23159  ;       with byte offset clear with a DATO to address 1FE first.  This checks
;23160  ;       the PAGE BOUND and TWO MEM CYC not branch.  Then the test is run with
;23161  ;       BYTE OFFSET set to address 1FE.  This checks the PAGE BOUND and TWO
;23162  ;       MEM CYC branch and also the extra states associated with writing across
;23163  ;       a page boundary.
;23164  ;         The UB MAP is set up to map virtual address 1FC to physical address
;23165  ;       1FC, and virtual address 200 to physical address 200.  This uses the
;23166  ;       first two locations in the UB MAP RAM.
;23167  ;
;23168  ; Assumptions:
;23169  ;
;23170  ;       It is assumed that all other micro-diagnostics have run successfully.
;23171  ;       It is also assumed that the UBE present is known to be good.
;23172  ;
;23173  ; Test Steps:
;23174  ;
;23175  ;       1)  Set up error mask, error number, and module number in LS (for
;23176  ;           error printouts) and clear previous error number in LS.  Also
;23177  ;           issue DCLO to UBE to clear error bits.
;23178  ;       2)  Set up UNIBUS MAP to map virtual address 0 to physical address 0
;23179  ;           and set the VALID bit.  Set up address 1 to map virtual 200 to
;23180  ;           physical 200 and set valid bit.
;23181  ;       3)  Clear the IPL bits in the PSW (allow all interrupts).  Write all
;23182  ;           zeros at addresses 1FC and 200.
;23183  ;       4)  Set up UBE data reg with all 1's, cycle count to 1 cycle (FFFF),
;23184  ;           and address reg to address 1FE.
;23185  ;       5)  Set up control reg 1 of the UBE to execute one DATO cycle via an
;23186  ;           NPR and interrupt via a BR6 when done.  Also set the GO bit to
;23187  ;           start the UBE.
;23188  ;       6)  Loop waiting for BR interrupt from UBE signifying that UBE is done.
;23189  ;       7)  Check the identifier code when BR interrupt occurs.  If a BR 7 was
;23190  ;           received (indicates a UBE error), read the UBE error register and
;23191  ;           report the error.
;23192  ;       8)  Read memory address 1FC and check for ones in bytes 2 and 3.
;23193  ;       9)  Read memory address 200 and check for all zeros (no change).
;23194  ;       10) Set up UNIBUS MAP to map virtual address 0 to physical address 0
;23195  ;           and set the VALID and BYTE OFFSET bits.  Set up address 1 to map
;23196  ;           virtual 200 to physical 200 and set valid and BYTE OFFSET bits.
;23197  ;       11) Write addresses 1FC and 200 with all zeros.  Repeat steps 4 thru 7.
;23198  ;       12) Read memory address 1FC and check for one byte of 1's in byte 3.
;23199  ;       13) Read memory address 200 and check for one byte of 1's in byte 0.
;23200  ;
;23201  ; Errors:
;23202  ;
;23203  ;       Error 1 - BR7 interrupt occurred when doing DATO (indicates UBE error).
;23204  ;                 Received data is contents of UBE control reg 2 (error reg-
;23205  ;                 ister).
;23206  ;       Error 2 - Main memory longword 1FC failed to get written correctly by
```

```
                ;23207  ;               UBE when writing a word to address 1FE.
                ;23208  ;       Error 3 - Main memory longword at address 200 modified by UBE write to
                ;23209  ;               address 1FE.
                ;23210  ;       Error 4 - BR7 interrupt occurred when doing DATO (indicates UBE error).
                ;23211  ;               Received data is contents of UBE control reg 2 (error reg-
                ;23212  ;               ister).
                ;23213  ;       Error 5 - Main memory longword 1FC failed to get written correctly by
                ;23214  ;               UBE when writing a word to address 1FE with byte offset set.
                ;23215  ;       Error 6 - Main memory longword 200 failed to get written correctly by
                ;23216  ;               UBE when writing a word to address 1FE with byte offset set.
                ;23217  ;
                ;23218  ; Debug:
                ;23219  ;
                ;23220  ;       Error 1 - This error indicates that the UBE detected an error.  See the
                ;23221  ;       UBE error register list in this listing to determine what went wrong.
                ;23222  ;
                ;23223  ;       Error 2 - This error most likely indicates a problem with the micro-
                ;23224  ;       code.  The micro-flow should take the PAGE BOUND and TWO MEM CYC not
                ;23225  ;       path at the cycle which issues the CSR ERR SUM CLK.  From there the
                ;23226  ;       UB ERRSUM not path to the cycle that issues SSYN should be taken.  The
                ;23227  ;       flow is the same as any 1 mem cyc DATO from that point.
                ;23228  ;
                ;23229  ;       Error 3 - Same as error 2.
                ;23230  ;
                ;23231  ;       Error 4 - This error indicates that the UBE detected an error.  See the
                ;23232  ;       UBE error register list in this listing to determine what went wrong.
                ;23233  ;
                ;23234  ;       Error 5 - This error most likely indicates a problem with the micro-
                ;23235  ;       code.  The micro-flow should take the PAGE BOUND and TWO MEM CYC path
                ;23236  ;       at the cycle that issues CSR ERR SUM CLK to UB CYC WR C5.  The UB ERRSUM
                ;23237  ;       not path should then be taken to a cycle which sets ICO for branching
                ;23238  ;       on later.  It then should go to UB CYC WR ARY C7 and take the normal
                ;23239  ;       2 MEM CYC flow up to the cycle which branches on ICO.  This time, the
                ;23240  ;       ICO not path should be taken.  The next cycle should branch on UB
                ;23241  ;       ERRSUM not to UB CYC WR ARY C17 and then to UB CYC WR ARY C18.  From
                ;23242  ;       there the flow is like the normal 2 MEM CYCLE exit.
                ;23243  ;
                ;23244  ;       Error 6 - Same as error 5.
                ;23245  ;
                ;23246  ;--
                ;23247
                ;23248  ;       T9  = ADDRESS 1FE FOR PAGE BOUNDARY
                ;23249  ;       T10 = ADDRESS 1FC TO WRITE TO LAST LONGWORD ON PAGE
                ;23250  ;       T14 = DATA FOR UBE CONTROL REG 1
                ;23251  ;       T15 = EXPECTED ARRAY DATA AT ADDRESS 0
                ;23252  ;       T57 = EXPECTED ARRAY DATA AT ADDRESS 4
                ;23253
                ;23254  T.3B:
U 1E0C, B65E,15 ;23255          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;23256                                          ; STATUS WORD
U 1E0D, 3E80,15 ;23257          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;23258                                          ; BIT 15 INDICATES START OF TEST TO
                ;23259                                          ; CONSOLE
U 1F0E, 10E0,15 ;23260          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;23261  WAIT.T3B.0:
```

```
U 1E0F, 89E0,F4 ;23262          JMP [WAIT.T3B.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;23263                                          ; RESPOND
                ;23264
U 1E10, 0A1B,8C ;23265          JSR [ISSUE.DCLO]                ;INIT UBE
U 1E11, 65A0,15 ;23266          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 1E12, 3644,15 ;23267          MOV LS[MCT] TO WR[0]            ;SET BIT 2 IN WR
U 1E13, 3E8C,15 ;23268          MOV WR[0] TO LS[MODULE.NUM]     ;SET UP TO CALL OUT MCT MODULE IF ERROR
                ;23269
U 1E14, B652,15 ;23270          MOV LS[#200] TO WR[0]           ;200 IN WR
U 1E15, 4142,15 ;23271          SUB LS[#2] FROM WR[0]           ;1FE IN WR
U 1E16, BE12,15 ;23272          MOV WR[0] TO LS[T9]             ;STORE 1FE IN LS
U 1E17, 4142,15 ;23273          SUB LS[#2] FROM WR[0]           ;1FC IN WR
U 1E18, BE14,15 ;23274          MOV WR[0] TO LS[T10]            ;STORE 1FC IN LS
                ;23275
U 1E19, 1B9D,F5 ;23276          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;23277                                          ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1E1A, B27E,15 ;23278          WRITE.MEM LS[BIT31]             ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;23279                                          ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                ;23280                                          ; OFFSET CLEAR
U 1E1B, 367E,15 ;23281          MOV LS[BIT31] TO WR[0]          ;SET BIT 31 IN WR
U 1E1C, C740,15 ;23282          BIS LS[#1] TO WR[0]             ;SET LSB OF WR 0
U 1E1D, 3E10,15 ;23283          MOV WR[0] TO LS[T8]             ;STORE IN LS
U 1E1E, 9B53,F5 ;23284          MEM.REQ[WRITE.UBS.MAP] ADRS[BIT9] DT[LONG]
                ;23285                                          ;SET UP TO WRITE ADDR 1 OF UNIBUS MAP
U 1E1F, 3210,15 ;23286          WRITE.MEM LS[T8]                ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;23287                                          ; ADDR 200 TO PHYSICAL ADDR 200, BYTE
                ;23288                                          ; OFFSET CLEAR
                ;23289
U 1E20, 2F80,15 ;23290          CLR WR[0]                       ;CLEAR WR 0
U 1E21, BFFE,15 ;23291          MOV WR[0] TO LS[PSL.HW]         ;CLEAR IPL BITS IN PSW
U 1E22, 0A1C,0C ;23292          JSR [SETUP.DATO.NPR]            ;SETUP T14 WITH UBE DATA TO ISSUE NPR
                ;23293                                          ; DATO, INTERRUPT ON DONE AT BR6
                ;23294
U 1E23, 3641,15 ;23295          MOV LS[#1] TO WR[2]             ;STARTING ERROR NUMBER (ERROR 1)
U 1E24, 9914,75 ;23296          MEM.REQ[WRITE.P] ADRS[T10] DT[LONG]
                ;23297                                          ;SETUP TO WRITE AT ADDRESS 1FC
U 1E25, B29C,15 ;23298          WRITE.MEM LS[ZERO]              ;WRITE 0'S AT ADDRESS 1FC
U 1E26, 1952,75 ;23299          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                ;23300                                          ;SETUP TO WRITE AT ADDRESS 200
U 1E27, B29C,15 ;23301          WRITE.MEM LS[ZERO]              ;WRITE 0'S AT ADDRESS 200
U 1E28, 19B4,35 ;23302          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                ;23303                                          ;SET UP TO WRITE TO UBE DATA REG
U 1E29, 329E,15 ;23304          WRITE.MEM LS[ONES]              ;PUT 1'S IN DATA REG
U 1E2A, 5F28,15 ;23305          MCOM LS[#FFFF] TO WR[0]         ;FFFF0000 IN WR
U 1E2B, BE1E,15 ;23306          MOV WR[0] TO LS[T15]            ;EXPECTED DATA IN FIRST LONGWORD
U 1E2C, E5AE,15 ;23307          CLR LS[T57]                     ;EXPECTED DATA IN SECOND LONGWORD
U 1E2D, 09E4,2C ;23308          JSR [LOOP.T3B.2.3]              ;PERFORM TEST WITH ALL 1'S DATA
                ;23309                                          ; WITH 1 MEMORY CYCLE
                ;23310
U 1E2E, 367E,15 ;23311          MOV LS[BIT31] TO WR[0]          ;SET BIT 31 IN WR
U 1E2F, 4772,15 ;23312          BIS LS[BIT25] TO WR[0]          ;SET BIT 25 (BYTE OFFSET)
U 1E30, 3E10,15 ;23313          MOV WR[0] TO LS[T8]             ;STORE IN LS
U 1E31, 1B9D,F5 ;23314          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;23315                                          ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1E32, 3210,15 ;23316          WRITE.MEM LS[T8]                ;SET VALID BIT IN UB MAP, MAP VIRTUAL
```

```
                 ;23317                                          ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                 ;23318                                          ; OFFSET SET
U 1E33, C740,15  ;23319          BIS LS[#1] TO WR[0]             ;SET LSB OF WR 0
U 1E34, 3E10,15  ;23320          MOV WR[0] TO LS[T8]             ;STORE IN LS
U 1E35, 9B53,F5  ;23321          MEM.REQ[WRITE.UBS.MAP] ADRS[BIT9] DT[LONG]
                 ;23322                                          ;SET UP TO WRITE ADDR 1 OF UNIBUS MAP
U 1E36, 3210,15  ;23323          WRITE.MEM LS[T8]                ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                 ;23324                                          ; ADDR 200 TO PHYSICAL ADDR 200, BYTE
                 ;23325                                          ; OFFSET SET
                 ;23326
U 1E37, B645,15  ;23327          MOV LS[#4] TO WR[2]             ;STARTING ERROR NUMBER (ERROR 4)
U 1E38, 9914,75  ;23328          MEM.REQ[WRITE.P] ADRS[T10] DT[LONG]
                 ;23329                                          ;SETUP TO WRITE AT ADDRESS 1FC
U 1E39, B29C,15  ;23330          WRITE.MEM LS[ZERO]              ;WRITE 0'S AT ADDRESS 1FC
U 1E3A, 1952,75  ;23331          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                 ;23332                                          ;SETUP TO WRITE AT ADDRESS 200
U 1E3B, B29C,15  ;23333          WRITE.MEM LS[ZERO]              ;WRITE 0'S AT ADDRESS 200
U 1E3C, B62A,15  ;23334          MOV LS[#FF000000] TO WR[0]      ;FF000000 IN WR
U 1E3D, BE1E,15  ;23335          MOV WR[0] TO LS[T15]            ;EXPECTED DATA AT ADDRESS 1FC
U 1E3E, B626,15  ;23336          MOV LS[#FF] TO WR[0]            ;FF IN WR
U 1E3F, 3EAE,15  ;23337          MOV WR[0] TO LS[T57]            ;EXPECTED DATA AT ADDRESS 200
U 1E40, 09E4,2C  ;23338          JSR [LOOP.T3B.2.3]              ;PERFORM TEST WITH ALL 1'S DATA
                 ;23339                                          ; WITH 2 MEM CYCLES
                 ;23340
U 1E41, 89E5,C4  ;23341          JMP [END.T3B]                   ;DONE
                 ;23342
                 ;23343  ;
                 ;23344  ;SUBROUTINE TO PERFORM NPR TEST
                 ;23345  ;
                 ;23346
                 ;23347  LOOP.T3B.2.3:
U 1E42, 3E83,15  ;23348          MOV WR[2] TO LS[ERROR.NUMBER]   ;SET UP ERR NUMBER IN LS (ERROR 1 OR 4)
                 ;23349  LOOP.T3B.1:
U 1E43, 19B8,35  ;23350          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                 ;23351                                          ;SET UP TO WRITE TO UBE ADDRESS REG
U 1E44, B212,15  ;23352          WRITE.MEM LS[T9]                ;PUT ADDRESS 1FE IN UBE ADDRESS REG
U 1E45, 99B6,35  ;23353          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                 ;23354                                          ;SET UP TO WRITE TO UBE CYCLE COUNT
U 1E46, 329E,15  ;23355          WRITE.MEM LS[ONES]              ;PUT A -1 IN CYCLE COUNT (DO 1 CYCLE)
U 1E47, 99BA,35  ;23356          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                 ;23357                                          ;SET UP TO WRITE TO UBE CONTROL REG 1
U 1E48, 321C,15  ;23358          WRITE.MEM LS[T14]               ;SET UP CONTROL REG TO ISSUE NPR TO
                 ;23359                                          ; WRITE A WORD TO ARRAY AND INTR AT
                 ;23360                                          ; BR6 ON DONE
                 ;23361  LOOP.T3B.INT.1:
U 1E49, 09E4,97  ;23362          JMP.IF[NO.INTERRUPT] TO [LOOP.T3B.INT.1] ;LOOP UNTIL BR INTERRUPT
                 ;23363
U 1E4A, 8A1C,DC  ;23364          JSR [CHECK.UBE.ERR]             ;SEE IF UBE ERR AND ISSUE NECESSARY
                 ;23365                                          ; GRANTS
U 1E4B, 09E4,34  ;23366          JMP [LOOP.T3B.1]                ;ERROR LOOP RETURN IF ENABLED
                 ;23367
U 1E4C, FF82,15  ;23368          INC LS[ERROR.NUMBER]            ;ERROR 2 OR 5
U 1E4D, E58A,15  ;23369          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
U 1E4E, 1815,F5  ;23370          MEM.REQ[READ.P] ADRS[T10] DT[LONG]
                 ;23371                                          ;SET UP TO READ DATA AT ADDRESS 1FC
```

```
U 1E4F, 3022,15 ;23372          MOV MEM.DATA TO WR[0]              ;GET RESULT
U 1E50, B61E,95 ;23373          MOV LS[T15] TO WR[1]               ;EXPECTED DATA
U 1E51, 0869,3C ;23374          JSR [CHECK.RESULT]                 ;SEE IF ARRAY WRITTEN CORRECTLY
U 1E52, 89E4,24 ;23375          JMP [LOOP.T3B.2.3]                 ;ERROR LOOP IF ENABLED
                ;23376
U 1E53, FF82,15 ;23377          INC LS[ERROR.NUMBER]               ;ERROR 3 OR 6
U 1E54, 9853,F5 ;23378          MEM.REQ[READ.P] ADRS[#200] DT[LONG]
                ;23379                                             ;SET UP TO READ DATA AT ADDRESS 200
U 1E55, 3022,15 ;23380          MOV MEM.DATA TO WR[0]              ;GET RESULT
U 1E56, 36AE,95 ;23381          MOV LS[T57] TO WR[1]               ;EXPECTED DATA
U 1E57, 0869,3C ;23382          JSR [CHECK.RESULT]                 ;SEE IF ARRAY WRITTEN CORRECTLY
U 1E58, 89E4,24 ;23383          JMP [LOOP.T3B.2.3]                 ;ERROR LOOP IF ENABLED
                ;23384
U 1E59, 99BC,35 ;23385          MEM.REQ[WRITE.P] ADRS[CLEAR.ERR.ADDR] DT[WORD]
                ;23386                                             ;SET UP TO CLEAR UBE ERROR REG
U 1E5A, B29C,15 ;23387          WRITE.MEM LS[ZERO]                 ;CLEAR UBE ERROR REG
U 1E5B, 5B00,14 ;23388          RETURN                             ;DONE WITH NPR TEST FOR THIS DATA
                ;23389
                ;23390  END.T3B:
```

```
;23391  .page   "  TEST 3C Partial UB CSR 2 and UB ERRSUM Test  (M8391 MCT Module)"
;23392  ;++
;23393  ;
;23394  ; Test Description:
;23395  ;
;23396  ;       This test checks the UB CSR 2 PAL logic for UB ERROR SUM, UB TB PAR
;23397  ;       ERR, UB NXM, and WR NOT VALID while doing a DATO write to the array.
;23398  ;       All possible error conditions that can occur on a UNIBUS WRITE that
;23399  ;       would cause an abort are checked in this test.
;23400  ;         The test first sets the TB PAR DIAG bit in CSR 1 and writes the UBS
;23401  ;       MAP.  The map thus contains a parity error.  The test tries to write
;23402  ;       to the array from the UBE at address 0 and address 1FE with byte
;23403  ;       offset both clear and then set.  This checks the generation and de-
;23404  ;       tection of UB ERRSUM at each of the four targets reached with PAGE
;23405  ;       BOUND and TWO MEM CYC in all four combinations.  The test will check
;23406  ;       that in all four cases, the NO SSYN ERR bit in UBE CONTROL REG 2 is
;23407  ;       set, the UB TB PAR ERR bit in UB CSR 2 (on the MCT module) is set, and
;23408  ;       that the write to the array was aborted.
;23409  ;         The next part of this test checks the UB ERRSUM branch in the 2 MEM
;23410  ;       CYC path by forcing a TB PARITY ERR in the map across a page boundary.
;23411  ;       The test should write the first part of the word (in location 1FF), but
;23412  ;       abort the second cycle write and not issue SSYN.  The UB TB PAR ERR in
;23413  ;       UB CSR 2 should also be set.
;23414  ;         WR NOT VALID is checked next by clearing the valid bit in the UB MAP
;23415  ;       in the location accessed across the page boundary.  The result should
;23416  ;       be similar to the UB TB PAR ERR in the second cycle above, except that
;23417  ;       WR NOT VALID is set in UB CSR 2.
;23418  ;         NXM is checked by mapping virtual address 0 to physical address
;23419  ;       500000 in the UBS MAP.  The NXM is set up to occur on the first cycle
;23420  ;       so the memory write should be aborted and the UB NXM bit should be
;23421  ;       set in UB CSR 2.
;23422  ;         Finally the VALID bit is cleared in the UBS MAP accessed first.  This
;23423  ;       causes an abort before a CSR ERR SUM clock is issued, so no bits should
;23424  ;       be set in UB CSR 2.
;23425  ;
;23426  ; Assumptions:
;23427  ;
;23428  ;       It is assumed that all other micro-diagnostics have run successfully.
;23429  ;       It is also assumed that the UBE present is known to be good.
;23430  ;
;23431  ; Test Steps:
;23432  ;
;23433  ;       1)  Set up error mask, error number, and module number in LS (for
;23434  ;           error printouts) and clear previous error number in LS.  Also
;23435  ;           issue DCLO to UBE to clear error bits.
;23436  ;       2)  Set up UNIBUS MAP to map virtual address 0 to physical address 0
;23437  ;           and set the VALID bit.  Set up address 1 to map virtual 200 to
;23438  ;           physical 200 and set VALID and BYTE OFFSET bits.  Write both
;23439  ;           map locations with forced bad TB parity.
;23440  ;       3)  Clear the IPL bits in the PSW (allow all interrupts).  Write all
;23441  ;           zeros at address 0.
;23442  ;       4)  Set up UBE data reg with all 1's, cycle count to 1 cycle (FFFF),
;23443  ;           and address reg to address 2.
;23444  ;       5)  Set up control reg 1 of the UBE to execute one DATO cycle via an
;23445  ;           NPR and interrupt via a BR6 when done.  Also set the GO bit to
```

```
;23446  ;          start the UBE.
;23447  ;       6) Loop waiting for BR interrupt from UBE signifying that UBE is done.
;23448  ;       7) Issue bus grant on level 6, then level 7 to clear BRs.
;23449  ;       8) Read UBE control reg 2 (error reg).  Check that NO SSYN ERR bit is
;23450  ;          set.
;23451  ;       9) Read memory address 0 and check that no write was executed.
;23452  ;       10) Read UB CSR 2 and check that UB TB PAR ERR is set.
;23453  ;       11) Repeat steps 4 through 10 above with UB MAP set up to execute 2 MEM
;23454  ;          CYCLES (BYTE OFFSET set) and cross a page boundary (use address 1FE)
;23455  ;       12) Repeat steps 4 through 10 with TB PAR ERROR in second map location.
;23456  ;          Check for first write (to 1FF) to complete, second write (to 200)
;23457  ;          aborted.
;23458  ;       13) Repeat steps 4 through 10 with UB MAP set up with VALID clear in
;23459  ;          second map location.  Check for UB WR NOT VALID in UB CSR 2 and
;23460  ;          write to 1FF to complete.
;23461  ;       14) Repeat steps 4 through 10 with UB MAP set up with physical address
;23462  ;          500000 to cause an NXM.  Check for NXM set in UB CSR 2.
;23463  ;       15) Repeat steps 4 through 10 with VALID clear in first UB MAP location.
;23464  ;          Check that no error bits in UB CSR 2 are written (no clock).
;23465  ;
;23466  ; Errors:
;23467  ;
;23468  ;       NOTE:  The data under OTHER is the address to which the UBE did the
;23469  ;              write.  Even addresses indicate BYTE OFFSET clear, odd addresses
;23470  ;              indicate BYTE OFFSET set.
;23471  ;
;23472  ;       Error 1 - UB CONTROL REG 2 did not have NO SSYN ERR set after DATO to
;23473  ;                 array with a UB TB ERR forced.
;23474  ;       Error 2 - Array address modified after DATO to array with UB TB PAR ERR
;23475  ;                 forced (write should have been aborted).
;23476  ;       Error 3 - UB TB PAR ERR bit in UB CSR 2 (on MCT module) not set after
;23477  ;                 DATO to array with UB TB PAR ERR forced.
;23478  ;       Error 4 - UB CONTROL REG 2 did not have NO SSYN ERR set after DATO to
;23479  ;                 array with a UB TB ERR forced in second cycle.
;23480  ;       Error 5 - First array address not written correctly after DATO to array
;23481  ;                 with UB TB PAR ERR in second cycle.
;23482  ;       Error 6 - UB TB PAR ERR bit in UB CSR 2 (on MCT module) not set after
;23483  ;                 DATO to array with UB TB PAR ERR forced in second cycle.
;23484  ;       Error 7 - Second cycle array address modified after DATO with UB TB PAR
;23485  ;                 ERR in that cycle (write should have aborted).  Address 200.
;23486  ;       Error 8 - UB CONTROL REG 2 did not have NO SSYN ERR set after DATO to
;23487  ;                 array with WR NOT VALID in second cycle.
;23488  ;       Error 9 - First array address not written correctly after DATO to array
;23489  ;                 with WR NOT VALID
;23490  ;       Error A - WR NOT VALID bit in UB CSR 2 (on MCT module) not set after
;23491  ;                 DATO to array with VALID clear in second cycle.
;23492  ;       Error B - UB CONTROL REG 2 did not have NO SSYN ERR set after DATO to
;23493  ;                 array with NXM.
;23494  ;       Error C - Array address modified after DATO to array with NXM error.
;23495  ;       Error D - NXM bit in UB CSR 2 (on MCT module) not set after DATO to
;23496  ;                 array with address 500000 in UB MAP.
;23497  ;       Error E - UB CONTROL REG 2 did not have NO SSYN ERR set after DATO to
;23498  ;                 array with VALID ERR in first cycle.
;23499  ;       Error F - Array address modified after DATO to array with valid error.
;23500  ;       Error 10- UB CSR 2 had an error bit set when no clock should have oc-
```

```
;23501  ;                   curred.
;23502  ;
;23503  ; Debug:
;23504  ;
;23505  ;       Error 1 - This error most likely indicates a problem with the micro-
;23506  ;       code or the UB ERRSUM logic.  If OTHER=2 (first check of UB ERRSUM),
;23507  ;       check the signal UB ERROR SUM L at the BEN 3 mux for a low after the
;23508  ;       cycle that issues CSR ERR SUM CLK in the UNIBUS WRITE TO ARRAY flow.
;23509  ;       If incorrect, check UB ERRSUM L at the output of the UB CSR2 PAL.  If
;23510  ;       incorrect there, check that CLR ERR L is high.  If not, this will pre-
;23511  ;       vent UB ERRSUM from asserting.  If CLR ERR L is high, the PAL is prob-
;23512  ;       ably bad.  If UB ERRSUM L is low, check that the micro-code branches to
;23513  ;       UB CYC ABORT WR CYC.  If not, either the BEN 3 mux is bad, or the micro
;23514  ;       code itself is failing.
;23515  ;         If OTHER is not equal to 2, then the most likely failure is the micro
;23516  ;       code itself.  OTHER indicates which path should be taken after the
;23517  ;       cycle that issues CSR ERR SUM CLK in the UNIBUS WRITE TO ARRAY flow.
;23518  ;       Each of these targets branches on UB ERRSUM, and should take the UB
;23519  ;       ERRSUM true path to UB CYC ABORT WR CYC.  OTHER=1FE takes the PAGE
;23520  ;       BOUND and TWO MEM CYC not path.  OTHER=3 takes the PAGE BOUND NOT and
;23521  ;       TWO MEM CYCLE path.  OTHER=1FF takes the PAGE BOUND and TWO MEM CYC
;23522  ;       path.
;23523  ;
;23524  ;       Error 2 - This error most likely will appear in conjunction with error
;23525  ;       1 and indicates that the micro-code did not exit on UB ERRSUM, but in-
;23526  ;       stead took the normal no error path and wrote to the array.  Debug is
;23527  ;       the same as for error 1.
;23528  ;
;23529  ;       Error 3 - If this is the first error, it most likely indicates that
;23530  ;       the UB CSR2 PAL is bad.
;23531  ;
;23532  ;       Error 4 - This error most likely indicates a problem with the micro
;23533  ;       code itself.  The micro code should have taken the UB ERRSUM path
;23534  ;       at the cycle in the 2 MEM CYC path that follows the CSR ERR SUM CLK,
;23535  ;       READ ARRAY, and WRITE SUBST cycle.
;23536  ;
;23537  ;       Error 5 - Same as error 4.
;23538  ;
;23539  ;       Error 6 - Same as error 4.  If this is the first error, suspect a bad
;23540  ;       UB CSR 2 PAL.
;23541  ;
;23542  ;       Error 7 - Same as error 4.
;23543  ;
;23544  ;       Error 8 - This error most likely indicates a problem with the UB CSR2
;23545  ;       PAL or the CSR 1B PAL.  Check the signal WR NOT VALID L going into the
;23546  ;       UB CSR2 PAL for a low after the VAR INCR cycle in the 2 MEM CYC path
;23547  ;       of the UNIBUS WRITE TO ARRAY flow.  If it is not low, check it at the
;23548  ;       output of the CSR 1B PAL.  Suspect the PAL if it is not low there.
;23549  ;         If WR NOT VALID L is low at the UB CSR2 PAL, this PAL is probably
;23550  ;       bad.
;23551  ;
;23552  ;       Error 9 - Same as error 8.
;23553  ;
;23554  ;       Error A - If this is the first error, suspect the UB CSR2 PAL.
;23555  ;
```

```
                    ;23556  ;       Error B - This error most likely indicates a problem with the UB CSR2
                    ;23557  ;       PAL or the input NXM L.  Check NXM L going into the UB CSR2 PAL after
                    ;23558  ;       the cycle that asserts CSR ERR SUM CLK for a low.  If incorrect, trace
                    ;23559  ;       it back to the DECODER B PAL.  If it is low, suspect the UB CSR2 PAL.
                    ;23560  ;
                    ;23561  ;       Error C - Same as error B.
                    ;23562  ;
                    ;23563  ;       Error D - Suspect the UB CSR2 PAL if this is the first error.
                    ;23564  ;
                    ;23565  ;       Error E - This error most likely indicates a problem with the micro
                    ;23566  ;       code or the branch on VALID logic.  Check the signal VALID H at the
                    ;23567  ;       BEN 0 mux for a high during UB CYC C2 which is the first cycle after
                    ;23568  ;       the dispatch on MSYN is taken.  If VALID H is high, suspect the BEN 0
                    ;23569  ;       mux or the micro code itself at UB CYC C2.  If VALID H is low, trace
                    ;23570  ;       it to its source at the TB RAM.
                    ;23571  ;
                    ;23572  ;       Error F - Same as error E.
                    ;23573  ;
                    ;23574  ;       Error 10 - Same as error E.
                    ;23575  ;
                    ;23576  ;--
                    ;23577
                    ;23578  ;       T9  = ADDRESS 1FE FOR PAGE BOUNDARY
                    ;23579  ;       T10 = ADDRESS 1FC TO WRITE TO LAST LONGWORD ON PAGE
                    ;23580  ;       T11 = ADDRESS TO WRITE IN UBE ADDRESS REG
                    ;23581  ;       T12 = ADDRESS TO CHECK IN MEMORY
                    ;23582  ;       T13 = EXPECTED DATA FOR FIRST CYCLE OF WRITE
                    ;23583  ;       T14 = DATA FOR UBE CONTROL REG 1
                    ;23584  ;       T57 = EXPECTED UB CSR 2 DATA (ERROR BITS)
                    ;23585
                    ;23586  T.3C:
U 1E5C, B65E,15     ;23587          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;23588                                          ; STATUS WORD
U 1E5D, 3E80,15     ;23589          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;23590                                          ; BIT 15 INDICATES START OF TEST TO
                    ;23591                                          ; CONSOLE
U 1E5E, 10E0,15     ;23592          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;23593  WAIT.T3C.0:
U 1E5F, 89E5,F4     ;23594          JMP [WAIT.T3C.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;23595                                          ; RESPOND
                    ;23596
U 1E60, 0A1B,8C     ;23597          JSR [ISSUE.DCLO]                ;INIT UBE
U 1E61, 65A0,15     ;23598          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 1E62, 3644,15     ;23599          MOV LS[MCT] TO WR[0]            ;SET BIT 2 IN WR
U 1E63, 3E8C,15     ;23600          MOV WR[0] TO LS[MODULE.NUM]     ;SET UP TO CALL OUT MCT MODULE IF ERROR
U 1E64, B670,15     ;23601          MOV LS[OTHER.DATA] TO WR[0]     ;SET OTHER DATA BIT IN WR
U 1E65, 3E80,15     ;23602          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP TO PRINT UNDER OTHER IF ERROR
                    ;23603
U 1E66, B652,15     ;23604          MOV LS[#200] TO WR[0]           ;200 IN WR
U 1E67, 4142,15     ;23605          SUB LS[#2] FROM WR[0]           ;1FE IN WR
U 1E68, BE12,15     ;23606          MOV WR[0] TO LS[T9]             ;STORE 1FE IN LS
U 1E69, 4142,15     ;23607          SUB LS[#2] FROM WR[0]           ;1FC IN WR
U 1E6A, BE14,15     ;23608          MOV WR[0] TO LS[T10]            ;STORE 1FC IN LS
                    ;23609
                    ;23610
```

```
U 1E6B, B67A,15 ;23611          MOV LS[BIT29] TO WR[0]              ;SET BIT 29 (TB PAR DIAG) IN WR
U 1E6C, 8A17,FC ;23612          JSR [WRITE.CSR1]                    ;SET TB PAR DIAG BIT IN CSR 1
U 1E6D, 1B9D,F5 ;23613          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;23614                                              ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1E6E, B27E,15 ;23615          WRITE.MEM LS[BIT31]                 ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;23616                                              ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                ;23617                                              ; OFFSET CLEAR, AND PARITY ERROR
U 1E6F, 367E,15 ;23618          MOV LS[BIT31] TO WR[0]              ;SET VALID IN WR 0
U 1E70, 4772,15 ;23619          BIS LS[BIT25] TO WR[0]              ;SET BYTE OFFSET ALSO
U 1E71, C740,15 ;23620          BIS LS[#1] TO WR[0]                 ;VIRTUAL 200 TO PHYSICAL 200
U 1E72, 3E10,15 ;23621          MOV WR[0] TO LS[T8]                 ;SAVE IN T8
U 1E73, 9B53,F5 ;23622          MEM.REQ[WRITE.UBS.MAP] ADRS[BIT9] DT[LONG]
                ;23623                                              ;SET UP TO WRITE ADDR 1 OF UNIBUS MAP
U 1E74, 3210,15 ;23624          WRITE.MEM LS[T8]                    ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;23625                                              ; ADDR 200 TO PHYSICAL ADDR 200, BYTE
                ;23626                                              ; OFFSET SET, PARITY ERROR PRESENT
U 1E75, 0A17,EC ;23627          JSR [CLEAR.CSR1]                    ;CLEAR CSR 1
                ;23628
U 1E76, 2F80,15 ;23629          CLR WR[0]                           ;CLEAR WR 0
U 1E77, BFFE,15 ;23630          MOV WR[0] TO LS[PSL.HW]              ;CLEAR IPL BITS IN PSW
U 1E78, 0A1C,0C ;23631          JSR [SETUP.DATO.NPR]                ;SETUP T14 WITH UBE DATA TO ISSUE NPR
                ;23632                                              ; DATO, INTERRUPT ON DONE AT BR6
U 1E79, 19B4,35 ;23633          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                ;23634                                              ;SET UP TO WRITE TO UBE DATA REG
U 1E7A, 329E,15 ;23635          WRITE.MEM LS[ONES]                  ;PUT 1'S IN DATA REG
                ;23636
U 1E7B, 3641,15 ;23637          MOV LS[#1] TO WR[2]                 ;STARTING ERROR NUMBER (ERROR 1)
U 1E7C, 8A1E,3C ;23638          JSR [CLEAR.CSR2]                    ;CLEAR ERRORS IN UB CSR 2
U 1E7D, 999C,75 ;23639          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;23640                                              ;SETUP TO WRITE AT ADDRESS 0
U 1E7E, B29C,15 ;23641          WRITE.MEM LS[ZERO]                  ;WRITE 0'S AT ADDRESS 0
U 1E7F, 3642,15 ;23642          MOV LS[#2] TO WR[0]                 ;PUT A 2 IN WR
U 1E80, 3E16,15 ;23643          MOV WR[0] TO LS[T11]                ;ADDRESS TO WRITE IN UBE ADDRESS REG
U 1E81, BE88,15 ;23644          MOV WR[0] TO LS[ADDRESS.DATA]       ;ADDRESS TO PRINT UNDER OTHER (2)
U 1E82, 6518,15 ;23645          CLR LS[T12]                         ;ADDRESS TO CHECK FOR NO WRITE
U 1E83, E51A,15 ;23646          CLR LS[T13]                         ;EXPECTED DATA IN ARRAY (WRITE ABORTED)
U 1E84, B65E,15 ;23647          MOV LS[BIT15] TO WR[0]              ;EXPECTED DATA IN UB CSR REG (UB TB PAR
                ;23648                                              ; ERR SET)
U 1E85, 3EAE,15 ;23649          MOV WR[0] TO LS[T57]                ;STORE IN LS
U 1E86, 09EF,5C ;23650          JSR [LOOP.T3C.3]                    ;PERFORM TEST WITH PAGE BOUND NOT AND
                ;23651                                              ; TWO MEM CYCLE NOT
                ;23652
U 1E87, 3641,15 ;23653          MOV LS[#1] TO WR[2]                 ;STARTING ERROR NUMBER (ERROR 1)
U 1E88, 8A1E,3C ;23654          JSR [CLEAR.CSR2]                    ;CLEAR ERRORS IN UB CSR 2
U 1E89, 9914,75 ;23655          MEM.REQ[WRITE.P] ADRS[T10] DT[LONG]
                ;23656                                              ;SETUP TO WRITE AT ADDRESS 1FC
U 1E8A, B29C,15 ;23657          WRITE.MEM LS[ZERO]                  ;WRITE 0'S AT ADDRESS 1FC
U 1E8B, 3612,15 ;23658          MOV LS[T9] TO WR[0]                 ;1FE IN WR
U 1E8C, BE88,15 ;23659          MOV WR[0] TO LS[ADDRESS.DATA]       ;ADDRESS TO PRINT UNDER OTHER IF ERROR
U 1E8D, 3E16,15 ;23660          MOV WR[0] TO LS[T11]                ;1FE IS ADDRESS TO WRITE IN UBE ADDR REG
U 1E8E, 3614,15 ;23661          MOV LS[T10] TO WR[0]                ;1FC IN WR
U 1E8F, BE18,15 ;23662          MOV WR[0] TO LS[T12]                ;ADDRESS TO CHECK FOR NO WRITE (1FC)
U 1E90, 09EF,5C ;23663          JSR [LOOP.T3C.3]                    ;PERFORM TEST WITH PAGE BOUND AND TWO
                ;23664                                              ; MEM CYCLES NOT
                ;23665
```

```
U 1E91, B67A,15 ;23666          MOV LS[BIT29] TO WR[0]           ;SET BIT 29 (TB PAR DIAG) IN WR
U 1E92, 8A17,FC ;23667          JSR [WRITE.CSR1]                 ;SET TB PAR DIAG BIT IN CSR 1
U 1E93, 367E,15 ;23668          MOV LS[BIT31] TO WR[0]           ;SET BIT 31 IN WR (VALID BIT)
U 1E94, 4772,15 ;23669          BIS LS[BIT25] TO WR[0]           ;SET BIT 25 (BYTE OFFSET)
U 1E95, 3E10,15 ;23670          MOV WR[0] TO LS[T8]              ;STORE IN LS
U 1E96, 1B9D,F5 ;23671          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;23672                                           ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1E97, 3210,15 ;23673          WRITE.MEM LS[T8]                 ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;23674                                           ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                ;23675                                           ; OFFSET SET, PARITY ERROR PRESENT
U 1E98, 0A17,EC ;23676          JSR [CLEAR.CSR1]                 ;CLEAR CSR1
                ;23677
U 1E99, 3641,15 ;23678          MOV LS[#1] TO WR[2]              ;STARTING ERROR NUMBER (ERROR 1)
U 1E9A, 8A1E,3C ;23679          JSR [CLEAR.CSR2]                 ;CLEAR ERRORS IN UB CSR 2
U 1E9B, 999C,75 ;23680          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;23681                                           ;SETUP TO WRITE AT ADDRESS 0
U 1E9C, B29C,15 ;23682          WRITE.MEM LS[ZERO]               ;WRITE 0'S AT ADDRESS 0
U 1E9D, 3642,15 ;23683          MOV LS[#2] TO WR[0]              ;PUT A 2 IN WR
U 1E9E, 3E16,15 ;23684          MOV WR[0] TO LS[T11]             ;ADDRESS TO WRITE IN UBE ADDRESS REG
U 1E9F, BE88,15 ;23685          MOV WR[0] TO LS[ADDRESS.DATA]    ;PUT A 2 IN OTHER TO START
U 1EA0, FF88,15 ;23686          INC LS[ADDRESS.DATA]             ;INC ADDRESS UNDER OTHER (ADDRESS 3)
U 1EA1, 6518,15 ;23687          CLR LS[T12]                      ;ADDRESS TO CHECK FOR NO WRITE
U 1EA2, 09EF,5C ;23688          JSR [LOOP.T3C.3]                 ;PERFORM TEST WITH PAGE BOUND NOT AND
                ;23689                                           ; TWO MEM CYCLES
                ;23690
U 1EA3, 3641,15 ;23691          MOV LS[#1] TO WR[2]              ;STARTING ERROR NUMBER (ERROR 1)
U 1EA4, 8A1E,3C ;23692          JSR [CLEAR.CSR2]                 ;CLEAR ERRORS IN UB CSR 2
U 1EA5, 9914,75 ;23693          MEM.REQ[WRITE.P] ADRS[T10] DT[LONG]
                ;23694                                           ;SETUP TO WRITE AT ADDRESS 1FC
U 1EA6, B29C,15 ;23695          WRITE.MEM LS[ZERO]               ;WRITE 0'S AT ADDRESS 1FC
U 1EA7, 3612,15 ;23696          MOV LS[T9] TO WR[0]              ;1FE IN WR
U 1EA8, BE88,15 ;23697          MOV WR[0] TO LS[ADDRESS.DATA]    ;PUT A 1FE IN LS TO START
U 1EA9, FF88,15 ;23698          INC LS[ADDRESS.DATA]             ;1FF FOR OTHER DATA
U 1EAA, 3E16,15 ;23699          MOV WR[0] TO LS[T11]             ;1FE IS ADDRESS TO WRITE IN UBE ADDR REG
U 1EAB, 3614,15 ;23700          MOV LS[T10] TO WR[0]             ;1FC IN WR
U 1EAC, BE18,15 ;23701          MOV WR[0] TO LS[T12]             ;ADDRESS TO CHECK FOR NO WRITE (1FC)
U 1EAD, 09EF,5C ;23702          JSR [LOOP.T3C.3]                 ;PERFORM TEST WITH PAGE BOUND AND TWO
                ;23703                                           ; MEM CYCLES
                ;23704
U 1EAE, 367E,15 ;23705          MOV LS[BIT31] TO WR[0]           ;SET BIT 31 IN WR (VALID BIT)
U 1EAF, 4772,15 ;23706          BIS LS[BIT25] TO WR[0]           ;SET BIT 25 (BYTE OFFSET)
U 1EB0, 3E10,15 ;23707          MOV WR[0] TO LS[T8]              ;STORE IN LS
U 1EB1, 1B9D,F5 ;23708          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;23709                                           ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1EB2, 3210,15 ;23710          WRITE.MEM LS[T8]                 ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;23711                                           ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                ;23712                                           ; OFFSET SET, NO PARITY ERROR
                ;23713
U 1EB3, B645,15 ;23714          MOV LS[#4] TO WR[2]              ;STARTING ERROR NUMBER (ERROR 4)
U 1EB4, 8A1E,3C ;23715          JSR [CLEAR.CSR2]                 ;CLEAR ERRORS IN UB CSR 2
U 1EB5, 9914,75 ;23716          MEM.REQ[WRITE.P] ADRS[T10] DT[LONG]
                ;23717                                           ;SETUP TO WRITE AT ADDRESS 1FC
U 1EB6, B29C,15 ;23718          WRITE.MEM LS[ZERO]               ;WRITE 0'S AT ADDRESS 1FC
U 1EB7, 1952,75 ;23719          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                ;23720                                           ;SETUP TO WRITE AT ADDRESS 200
```

```
U 1EB8, B29C,15 ;23721          WRITE.MEM LS[ZERO]                ;WRITE 0'S AT ADDRESS 200
U 1EB9, 3612,15 ;23722          MOV LS[T9] TO WR[0]               ;1FE IN WR
U 1EBA, BE88,15 ;23723          MOV WR[0] TO LS[ADDRESS.DATA]     ;PUT A 1FF IN LS TO START
U 1EBB, FF88,15 ;23724          INC LS[ADDRESS.DATA]              ;1FF FOR OTHER DATA
U 1EBC, 3E16,15 ;23725          MOV WR[0] TO LS[T11]              ;1FE IS ADDRESS TO WRITE IN UBE ADDR REG
U 1EBD, 3614,15 ;23726          MOV LS[T10] TO WR[0]              ;1FC IN WR
U 1EBE, BE18,15 ;23727          MOV WR[0] TO LS[T12]              ;ADDRESS TO CHECK FOR WRITE (1FC)
U 1EBF, B62A,15 ;23728          MOV LS[#FF000000] TO WR[0]        ;EXPECTED DATA IN 1FC
U 1EC0, 3E1A,15 ;23729          MOV WR[0] TO LS[T13]              ;SAVE IN LS
U 1EC1, 09EF,5C ;23730          JSR [LOOP.T3C.3]                  ;PERFORM TEST WITH PAGE BOUND AND TWO
                ;23731                                            ; MEM CYCLES WITH PARITY ERROR IN
                ;23732                                            ; SECOND CYCLE (ACROSS PAGE)
                ;23733  T3C.7:
U 1EC2, FF82,15 ;23734          INC LS[ERROR.NUMBER]              ;ERROR 7
U 1EC3, E58A,15 ;23735          CLR LS[ERROR.MASK]                ;CHECK ALL BITS
U 1EC4, 2F82,95 ;23736          CLR WR[1]                         ;EXPECTED DATA
U 1EC5, 9853,F5 ;23737          MEM.REQ[READ.P] ADRS[#200] DT[LONG]
                ;23738                                            ;SET UP TO READ ADDRESS 200
U 1EC6, 3022,15 ;23739          MOV MEM.DATA TO WR[0]             ;GET DATA
U 1EC7, 0869,3C ;23740          JSR [CHECK.RESULT]                ;MAKE SURE NO WRITE ON SECOND CYCLE
U 1EC8, 5B00,1E ;23741          SKIP                              ;LOOP ON ERROR IF ENABLED (SKIP JUMP
                ;23742                                            ; TO NEXT ERROR)
U 1EC9, 89EC,C4 ;23743          JMP [T3C.8]                       ;GO TO NEXT ERROR
U 1ECA, 89EF,4C ;23744          JSR [LOOP.T3C.7]                  ;ERROR LOOP IF ENABLED
U 1ECB, 09EC,24 ;23745          JMP [T3C.7]                       ;REPEAT THIS ERROR
                ;23746
                ;23747  T3C.8:
U 1ECC, 3672,15 ;23748          MOV LS[BIT25] TO WR[0]            ;SET BIT 25 (BYTE OFFSET)
U 1ECD, C740,15 ;23749          BIS LS[#1] TO WR[0]
U 1ECE, 3E10,15 ;23750          MOV WR[0] TO LS[T8]               ;STORE IN LS
U 1ECF, 9B53,F5 ;23751          MEM.REQ[WRITE.UBS.MAP] ADRS[BIT9] DT[LONG]
                ;23752                                            ;SET UP TO WRITE ADDR 1 OF UNIBUS MAP
U 1ED0, 3210,15 ;23753          WRITE.MEM LS[T8]                  ;CLEAR VALID BIT IN UB MAP, MAP VIRTUAL
                ;23754                                            ; ADDR 200 TO PHYSICAL ADDR 200, BYTE
                ;23755                                            ; OFFSET SET, NO PARITY ERROR
                ;23756
U 1ED1, 3647,15 ;23757          MOV LS[#8] TO WR[2]               ;STARTING ERROR NUMBER (ERROR 8)
U 1ED2, 8A1E,3C ;23758          JSR [CLEAR.CSR2]                  ;CLEAR ERRORS IN UB CSR 2
U 1ED3, 9914,75 ;23759          MEM.REQ[WRITE.P] ADRS[T10] DT[LONG]
                ;23760                                            ;SETUP TO WRITE AT ADDRESS 1FC
U 1ED4, B29C,15 ;23761          WRITE.MEM LS[ZERO]                ;WRITE 0'S AT ADDRESS 1FC
U 1ED5, 1952,75 ;23762          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                ;23763                                            ;SETUP TO WRITE AT ADDRESS 200
U 1ED6, B29C,15 ;23764          WRITE.MEM LS[ZERO]                ;WRITE 0'S AT ADDRESS 200
U 1ED7, 365C,15 ;23765          MOV LS[BIT14] TO WR[0]            ;EXPECTED DATA IN UB CSR 2 (WR NOT VALID)
U 1ED8, 3EAE,15 ;23766          MOV WR[0] TO LS[T57]              ;STORE IN LS
                ;23767                                            ;OTHER LS LOCATIONS ARE SAME AS PRE-
                ;23768                                            ; VIOUS ERROR
U 1ED9, 09EF,5C ;23769          JSR [LOOP.T3C.3]                  ;PERFORM TEST WITH PAGE BOUND AND TWO
                ;23770                                            ; MEM CYCLES WITH VALID ERROR IN SECOND
                ;23771                                            ; CYCLE (ACROSS PAGE)
                ;23772
U 1EDA, 367E,15 ;23773          MOV LS[BIT31] TO WR[0]            ;SET VALID BIT
U 1EDB, 4756,15 ;23774          BIS LS[BIT11] TO WR[0]            ;SET PA 21
U 1EDC, 475A,15 ;23775          BIS LS[BIT13] TO WR[0]            ;SET PA 23
```

```
U 1EDD, 3E10,15 ;23776          MOV WR[0] TO LS[T8]              ;STORE IN LS
U 1EDE, 1B9D,F5 ;23777          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;23778                                          ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1EDF, 3210,15 ;23779          WRITE.MEM LS[T8]                ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;23780                                          ; ADDR 0 TO PHYSICAL ADDR 500000, BYTE
                ;23781                                          ; OFFSET CLR, NO PARITY ERROR
                ;23782
U 1EE0, 40C1,15 ;23783          ADD LS[#3(H)] TO WR[2]          ;STARTING ERROR NUMBER (ERROR B)
U 1EE1, 8A1E,3C ;23784          JSR [CLEAR.CSR2]                ;CLEAR ERRORS IN UB CSR 2
U 1EE2, 999C,75 ;23785          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;23786                                          ;SETUP TO WRITE AT ADDRESS 0
U 1EE3, B29C,15 ;23787          WRITE.MEM LS[ZERO]              ;WRITE 0'S AT ADDRESS 0
U 1EE4, 6588,15 ;23788          CLR LS[ADDRESS.DATA]            ;ADDRESS TO PRINT UNDER OTHER IF ERROR
U 1EE5, E516,15 ;23789          CLR LS[T11]                     ;0 IS ADDRESS TO WRITE IN UBE ADDR REG
U 1EE6, 6518,15 ;23790          CLR LS[T12]                     ;ADDRESS TO CHECK FOR WRITE 0
U 1EE7, E51A,15 ;23791          CLR LS[T13]                     ;EXPECTED DATA IN ARRAY
U 1EE8, 3660,15 ;23792          MOV LS[BIT16] TO WR[0]          ;EXPECTED DATA IN UB CSR 2 (UB NXM)
U 1EE9, 3EAE,15 ;23793          MOV WR[0] TO LS[T57]            ;STORE IN LS
U 1EEA, 09EF,5C ;23794          JSR [LOOP.T3C.3]                ;PERFORM TEST WITH PAGE BOUND NOT AND
                ;23795                                          ; TWO MEM CYCLES NOT WITH ADDRESS TO
                ;23796                                          ; PRODUCE NXM (ADDR 500000)
                ;23797
U 1EEB, 1B9D,F5 ;23798          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;23799                                          ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1EEC, B29C,15 ;23800          WRITE.MEM LS[#0]                ;CLEAR VALID BIT IN UB MAP, MAP VIRTUAL
                ;23801                                          ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                ;23802                                          ; OFFSET CLR, NO PARITY ERROR
                ;23803
U 1EED, 40C1,15 ;23804          ADD LS[#3(H)] TO WR[2]          ;STARTING ERROR NUMBER (ERROR E)
U 1EEE, 8A1E,3C ;23805          JSR [CLEAR.CSR2]                ;CLEAR ERRORS IN UB CSR 2
U 1EEF, 999C,75 ;23806          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;23807                                          ;SETUP TO WRITE AT ADDRESS 0
U 1EF0, B29C,15 ;23808          WRITE.MEM LS[ZERO]              ;WRITE 0'S AT ADDRESS 0
U 1EF1, E5AE,15 ;23809          CLR LS[T57]                     ;EXPECTED DATA IN UB CSR (NO VALID IN
                ;23810                                          ; FIRST CYCLE DOESN'T WRITE UB CSR 2)
U 1EF2, 09EF,5C ;23811          JSR [LOOP.T3C.3]                ;PERFORM TEST WITH PAGE BOUND NOT AND
                ;23812                                          ; TWO MEM CYCLES NOT WITH VALID CLEAR
                ;23813                                          ; IN FIRST CYCLE
U 1EF3, 89F1,A4 ;23814          JMP [END.T3C]                   ;DONE
                ;23815
                ;23816  ;
                ;23817  ;SUBROUTINE TO PERFORM NPR TEST
                ;23818  ;
                ;23819  LOOP.T3C.7:
U 1EF4, FC88,15 ;23820          DEC LS[ADDRESS.DATA]            ;RESTORE OTHER TO 1FF FOR ERROR 7 LOOP
                ;23821  LOOP.T3C.3:
U 1EF5, E58A,15 ;23822          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
                ;23823  LOOP.T3C.2:
U 1EF6, 3E83,15 ;23824          MOV WR[2] TO LS[ERROR.NUMBER]   ;SET UP ERR NUMBER IN LS (ERRORS 1, 4,
                ;23825                                          ; 8, B, E)
                ;23826  LOOP.T3C.1:
U 1EF7, 19B8,35 ;23827          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                ;23828                                          ;SET UP TO WRITE TO UBE ADDRESS REG
U 1EF8, 3216,15 ;23829          WRITE.MEM LS[T11]               ;PUT ADDRESS IN UBE ADDRESS REG
U 1EF9, 99B6,35 ;23830          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
```

```
                 ;23831                                          ;SET UP TO WRITE TO UBE CYCLE COUNT                    U
U 1EFA, 329E,15  ;23832          WRITE.MEM LS[ONES]              ;PUT A -1 IN CYCLE COUNT (DO 1 CYCLE)                  U
U 1EFB, 99BA,35  ;23833          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]                                                  U
                 ;23834                                          ;SET UP TO WRITE TO UBE CONTROL REG 1
U 1EFC, 321C,15  ;23835          WRITE.MEM LS[T14]               ;SET UP CONTROL REG TO ISSUE NPR TO                    U
                 ;23836                                          ; WRITE A WORD TO ARRAY AND INTR AT
                 ;23837                                          ; BR6 ON DONE                                          U
                 ;23838  LOOP.T3C.INT.1:                                                                                U
U 1EFD, 09EF,D7  ;23839          JMP.IF[NO.INTERRUPT] TO [LOOP.T3C.INT.1] ;LOOP UNTIL BR INTERRUPT
                 ;23840
U 1EFE, 1F46,75  ;23841          MEM.REQ[ISSUE.BG] ADRS[BG6] DT[LONG]
                 ;23842                                          ;SET UP TO ISSUE BUS GRANT ON LEVEL 6
U 1EFF, 3022,15  ;23843          MOV MEM.DATA TO WR[0]           ;COMPLETE CYCLE
U 1F00, DB00,15  ;23844          NOP                             ;DELAY FOR NEXT INTR AT BR7
U 1F01, 1FCE,75  ;23845          MEM.REQ[ISSUE.BG] ADRS[BG7] DT[LONG]
                 ;23846                                          ;SET UP TO ISSUE BUS GRANT ON LEVEL 7
U 1F02, 3022,15  ;23847          MOV MEM.DATA TO WR[0]           ;COMPLETE CYCLE
                 ;23848
U 1F03, B650,95  ;23849          MOV LS[NO.SSYN.ERR] TO WR[1]    ;EXPECTED DATA (NO SSYN ERR IN UBE)
U 1F04, C75A,95  ;23850          BIS LS[CCOVF] TO WR[1]          ;ALSO CCOVF WILL BE SET
U 1F05, 98BF,B5  ;23851          MEM.REQ[READ.P] ADRS[BECR2] DT[WORD]
                 ;23852                                          ;SET UP TO READ UBE ERROR REG
U 1F06, 3022,15  ;23853          MOV MEM.DATA TO WR[0]           ;GET CONTENTS
U 1F07, 0869,3C  ;23854          JSR [CHECK.RESULT]              ;CHECK FOR 'NO SSYN ERR' BIT SET
U 1F08, 09EF,74  ;23855          JMP [LOOP.T3C.1]                ;ERROR LOOP IF ENABLED
                 ;23856
U 1F09, FF82,15  ;23857          INC LS[ERROR.NUMBER]            ;ERROR 2, 5, 9, C, F
U 1F0A, 1819,F5  ;23858          MEM.REQ[READ.P] ADRS[T12] DT[LONG]
                 ;23859                                          ;SET UP TO READ DATA
U 1F0B, 3022,15  ;23860          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1F0C, 361A,95  ;23861          MOV LS[T13] TO WR[1]            ;EXPECTED DATA
U 1F0D, 0869,3C  ;23862          JSR [CHECK.RESULT]              ;SEE IF ARRAY WRITE WAS ABORTED
U 1F0E, 89EF,64  ;23863          JMP [LOOP.T3C.2]                ;ERROR LOOP IF ENABLED
                 ;23864
U 1F0F, FF82,15  ;23865          INC LS[ERROR.NUMBER]            ;ERROR 3, 6, A, D, 10
U 1F10, 3630,15  ;23866          MOV LS[CSR2.MASK] TO WR[0]      ;MASK TO CHECK ERROR BITS IN CSR 2
U 1F11, 3E8A,15  ;23867          MOV WR[0] TO LS[ERROR.MASK]     ;SET UP ERROR MASK TO CHECK ERROR BITS
U 1F12, 36AE,95  ;23868          MOV LS[T57] TO WR[1]            ;EXPECTED DATA
U 1F13, 1D47,75  ;23869          MEM.REQ[READ.CSR] ADRS[CSR2] DT[LONG]
                 ;23870                                          ;SET UP TO READ UB CSR 2
U 1F14, 3022,15  ;23871          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1F15, 0869,3C  ;23872          JSR [CHECK.RESULT]              ;CHECK FOR UB TB PARITY ERROR SET
U 1F16, 89EF,54  ;23873          JMP [LOOP.T3C.3]                ;ERROR LOOP IF ENABLED
                 ;23874
U 1F17, 99BC,35  ;23875          MEM.REQ[WRITE.P] ADRS[CLEAR.ERR.ADDR] DT[WORD]
                 ;23876                                          ;SET UP TO CLEAR UBE ERROR REG
U 1F18, B29C,15  ;23877          WRITE.MEM LS[ZERO]              ;CLEAR UBE ERROR REG
U 1F19, 5B00,14  ;23878          RETURN                          ;DONE WITH NPR TEST FOR THIS DATA
                 ;23879
                 ;23880  END.T3C:
```

```
;23881  .page   "  TEST 3D UBE Read To Array Test  (M8391 MCT Module)"
;23882  ;++
;23883  ;
;23884  ; Test Description:
;23885  ;
;23886  ;       This test checks the logic and micro-flow associated with the UNIBUS
;23887  ;       READ TO ARRAY routine.  The is the first test to do a read initiated
;23888  ;       by the UBE
;23889  ;         The test sets up the UBE to do a READ TO ARRAY.  The UBE first as-
;23890  ;       serts NPR to the MCT, and the MCT responds with NPG.  The UBE will as-
;23891  ;       sert SACK when it sees NPG.  The MCT will then drop NPG.  The UBE will
;23892  ;       assert BBSY and negate SACK.  The UBE can now issue MSYN and begin the
;23893  ;       transfer cycle.
;23894  ;         Now the data path is as follows.  After the UBE issues MSYN,  the MCT
;23895  ;       exits from the idle loop taking the MSYN path to UB CYC C2.  The UB
;23896  ;       area of the TB RAMS is selected and REF and VALID are checked.  The
;23897  ;       microcode takes the VALID path to UB CYC C3A (VALID comes from the
;23898  ;       UNIBUS MAP area of the TB RAMS).  At UB CYC C3A the data latches for
;23899  ;       UB data are opened and the direction set up in preperation for a write
;23900  ;       to array.  A branch on UBC1 and MSYN is performed.  The microcode takes
;23901  ;       the MSYN and UBC1 not path (UBC1 is low since this is a DATI from the
;23902  ;       UBE) to UB CYC RD C4.  Here the UB DIR LATCH is disabled so the array
;23903  ;       can drive the UB DATA lines later.  A CSR ERR SUM CLK is issued, a READ
;23904  ;       ARRAY is executed, and a branch on 2 MEM CYC is executed.  The rest of
;23905  ;       this flow applies to a 1 MEM CYC read to the UBE.
;23906  ;         The micro flow takes the 2 MEM CYC not path to the next cycle, which
;23907  ;       continues the READ ARRAY, opens the ECC latches, and enables MDR data
;23908  ;       out.  The following cycle repeats the previos cycle and also enables
;23909  ;       the UB DATA onto the UNIBUS data lines.  A STALL MBSY is asserted also
;23910  ;       to prevent the issue of SSYN if a data error is detected.  The cycle
;23911  ;       will branch on UB ERRSUM not to the next cycle.  This cycle enables
;23912  ;       the data onto the UNIBUS data lines again and issues SSYN if no data
;23913  ;       error was detected.  It branches on ERROR.
;23914  ;         The micro code will take the ERROR not path and repeats the previous
;23915  ;       cycle except there is no STALL MBSY.  The following cycle loops waiting
;23916  ;       for MSYN to go away.
;23917  ;         After the UBE receives the SSYN issued by the MCT, the UBE will drop
;23918  ;       MSYN.  When the MCT receives the negation of MSYN, it will drop SSYN.
;23919  ;       The UBE will also drop BBSY, after a short delay, when it receives
;23920  ;       SSYN.  The MCT will deassert UB ACTIVITY when it sees BBSY go away.
;23921  ;         After MSYN goes away, the micro code goes to the next cycle which
;23922  ;       waits for UB ACTIVE to drop.  This occurs when the UBE negates BBSY.
;23923  ;       The micro code will then return to IDLE.
;23924  ;         Data patterns used are all 1's and all 0's.
;23925  ;
;23926  ; Assumptions:
;23927  ;
;23928  ;       It is assumed that all other micro-diagnostics have run successfully.
;23929  ;       It is also assumed that the UBE present is known to be good.
;23930  ;
;23931  ; Test Steps:
;23932  ;
;23933  ;       1)  Set up error mask, error number, and module number in LS (for
;23934  ;           error printouts) and clear previous error number in LS.  Also
;23935  ;           issue DCLO to UBE to clear error bits.
```

```
;23936  ;       2)  Set up UNIBUS MAP to map virtual address 0 to physical address 0
;23937  ;           and set the VALID bit.
;23938  ;       3)  Clear the IPL bits in the PSW (allow all interrupts).  Write all
;23939  ;           1's at physical memory address 0 and all 0's at address 4.
;23940  ;       4)  Set up UBE cycle count to 1 cycle (FFFF), and address reg to ad-
;23941  ;           dress 2.
;23942  ;       5)  Set up control reg 1 of the UBE to execute one DATI cycle via an
;23943  ;           NPR and interrupt via a BR6 when done.  Also set the GO bit to
;23944  ;           start the UBE.
;23945  ;       6)  Wait for BR interrupt and check the identifier code when interrupt
;23946  ;           occurs.  If a BR 7 was received (indicates a UBE error), read the
;23947  ;           UBE error register and report the error.
;23948  ;       7)  Issue BG to clear BR request.
;23949  ;       8)  Read UBE data reg and check for 1's (data read from memory).
;23950  ;       9)  Change address reg to address 4.  Repeat steps 5 through 8 checking
;23951  ;           for all 0's in step 8.
;23952  ;       10) Set the BYTE OFFSET bit in the UB MAP and repeat steps 4 through 8
;23953  ;           checking for 1's in low byte and 0's in high byte in step 8.
;23954  ;
;23955  ; Errors:
;23956  ;
;23957  ;       Note:  Data under OTHER in error report is physical address that the
;23958  ;       UBE is trying to read from main memory.
;23959  ;
;23960  ;       Error 1 - BR7 interrupt occurred (indicates UBE error).  Received data
;23961  ;                 is contents of UBE control reg 2 (error register).
;23962  ;       Error 2 - Main memory failed to get read correctly by UBE.  Address
;23963  ;                 under OTHER.
;23964  ;
;23965  ; Debug:
;23966  ;
;23967  ;       Error 1 - This error indicates that the UBE detected an error.  See the
;23968  ;       UBE error register list in this listing to determine what went wrong.
;23969  ;
;23970  ;       Error 2 - This error indicates that the UBE failed to read data cor-
;23971  ;       rectly from the array.  The most likely problem is with the micro-code
;23972  ;       itself, although the DATA ROTATOR CONTROL PAL is a possibility (see
;23973  ;       below).  The micro-code should take the LUBC1 not path at UB CYC C3A
;23974  ;       to UB CYC RD C4.  If not, check that L UBC1 H is high going into the
;23975  ;       BEN 1 MUX.  If it is, the microcode is probably bad.  The rest of the
;23976  ;       micro-code can be traced if the L UBC1 not branch is working correctly.
;23977  ;       The description above under TEST DESCRIPTION is for 1 MEM CYC reads
;23978  ;       (OTHER=2 or 4).  For 2 MEM CYC reads, see OTHER=3 below for correct
;23979  ;       micro-flow.
;23980  ;
;23981  ;       OTHER=2 or 4
;23982  ;
;23983  ;         If the result is shifted or only one byte is written, suspect the
;23984  ;       DATA ROTATOR CONTROL PAL.  Check that 2 MEM CYCLES L is high during the
;23985  ;       cycle that asserts ROT CLOCK H (UB CYC C3A).  If not, the PAL is prob-
;23986  ;       ably bad.
;23987  ;
;23988  ;       OTHER=3
;23989  ;
;23990  ;         If the result is shifted or only one byte is written, suspect the
```

```
                    ;23991  ;       DATA ROTATOR CONTROL PAL.  Check that 2 MEM CYCLES L is low during the
                    ;23992  ;       cycle that asserts ROT CLOCK H (UB CYC C3A).  If not, the PAL is prob-
                    ;23993  ;       ably bad.
                    ;23994  ;         The micro-flow for this read is as follows: UB CYC RD C4 to UB CYC RD
                    ;23995  ;       C5 to UB CYC TWO ARY CYC RD C6 to UB CYC TWO ARY CYC RD C7 to UB CYC
                    ;23996  ;       TWO ARY CYC RD C8 to unnamed cycle to UB CYC RD C4.  Then to UB CYC RD
                    ;23997  ;       C5 along the 2ND CYC path and then to UB CYC RD C7.  From there to IDLE
                    ;23998  ;       is the same as description under Test Description.
                    ;23999  ;
                    ;24000  ;--
                    ;24001
                    ;24002  ;       T14 = DATA FOR UBE CONTROL REG 1
                    ;24003  ;       T15 = EXPECTED ARRAY DATA
                    ;24004  ;       T58 = ADDRESS TO WRITE IN UBE ADDRESS REG
                    ;24005
                    ;24006  T.3D:
U 1F1A, B65E,15     ;24007          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;24008                                          ; STATUS WORD
U 1F1B, 3E80,15     ;24009          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;24010                                          ; BIT 15 INDICATES START OF TEST TO
                    ;24011                                          ; CONSOLE
U 1F1C, 10E0,15     ;24012          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;24013  WAIT.T3D.0:
U 1F1D, 09F1,D4     ;24014          JMP [WAIT.T3D.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;24015                                          ; RESPOND
                    ;24016
U 1F1E, 0A1B,8C     ;24017          JSR [ISSUE.DCLO]                ;INIT UBE
U 1F1F, 65A0,15     ;24018          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 1F20, 3644,15     ;24019          MOV LS[MCT] TO WR[0]            ;SET BIT 2 IN WR
U 1F21, 3E8C,15     ;24020          MOV WR[0] TO LS[MODULE.NUM]      ;SET UP TO CALL OUT MCT MODULE IF ERROR
U 1F22, 5F28,15     ;24021          MCOM LS[#FFFF] TO WR[0]         ;CLEAR BITS 15-00
U 1F23, 3E8A,15     ;24022          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK BITS 15-00
U 1F24, 1B9D,F5     ;24023          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                    ;24024                                          ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1F25, B27E,15     ;24025          WRITE.MEM LS[BIT31]             ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                    ;24026                                          ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                    ;24027                                          ; OFFSET CLEAR
U 1F26, 2F80,15     ;24028          CLR WR[0]                       ;CLEAR WR 0
U 1F27, BFFE,15     ;24029          MOV WR[0] TO LS[PSL.HW]          ;CLEAR IPL BITS IN PSW
U 1F28, 999C,75     ;24030          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                    ;24031                                          ;SET UP TO WRITE AT MEMORY ADDR 0
U 1F29, 329E,15     ;24032          WRITE.MEM LS[ONES]              ;WRITE ALL 1'S AT ADDR 0
U 1F2A, 9944,75     ;24033          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                    ;24034                                          ;SET UP TO WRITE AT MEMORY ADDR 4
U 1F2B, B29C,15     ;24035          WRITE.MEM LS[ZERO]              ;WRITE ALL 0'S AT ADDR 4
                    ;24036
U 1F2C, 8A1C,7C     ;24037          JSR [SETUP.DATI.NPR]            ;SETUP T14 WITH UBE DATA TO ISSUE NPR
                    ;24038                                          ; DATI, INTERRUPT ON DONE AT BR6
                    ;24039
U 1F2D, B670,15     ;24040          MOV LS[OTHER.DATA] TO WR[0]     ;SET BIT TO PRINT UNDER OTHER DATA
U 1F2E, 3E80,15     ;24041          MOV WR[0] TO LS[CONTROL.STATUS] ;WRITE IN CONTROL AND STATUS WORD
                    ;24042
U 1F2F, 3642,15     ;24043          MOV LS[#2] TO WR[0]             ;2 IN WR
U 1F30, BE88,15     ;24044          MOV WR[0] TO LS[ADDRESS.DATA]   ;OTHER DATA IS 2 (ADDRESS 2)
U 1F31, 3EB0,15     ;24045          MOV WR[0] TO LS[T58]            ;DATA OF 2 TO WRITE IN UBE ADDRESS REG
```

```
U 1F32, B69E,15 ;24046          MOV LS[ONES] TO WR[0]
U 1F33, BE1E,15 ;24047          MOV WR[0] TO LS[T15]             ;EXPECTED DATA (ALL 1'S)
U 1F34, 09F4,6C ;24048          JSR [LOOP.T3D.2]                 ;PERFORM TEST WITH ALL 1'S DATA
                ;24049                                           ; TO ADDRESS 2
                ;24050
U 1F35, 3644,15 ;24051          MOV LS[#4] TO WR[0]              ;4 IN WR
U 1F36, BE88,15 ;24052          MOV WR[0] TO LS[ADDRESS.DATA]    ;OTHER DATA IS 4 (ADDRESS 4)
U 1F37, 3EB0,15 ;24053          MOV WR[0] TO LS[T58]             ;DATA OF 4 TO WRITE IN UBE ADDRESS REG
U 1F38, 651E,15 ;24054          CLR LS[T15]                      ;EXPECTED DATA (ALL 0'S)
U 1F39, 09F4,6C ;24055          JSR [LOOP.T3D.2]                 ;PERFORM TEST WITH ALL 0'S DATA
                ;24056                                           ; TO ADDRESS 4
                ;24057
U 1F3A, 367E,15 ;24058          MOV LS[BIT31] TO WR[0]           ;SET BIT 31 IN WR (VALID BIT)
U 1F3B, 4772,15 ;24059          BIS LS[BIT25] TO WR[0]           ;SET BIT 25 IN WR (BYTE OFFSET)
U 1F3C, 3E10,15 ;24060          MOV WR[0] TO LS[T8]              ;STORE IN LS
U 1F3D, 1B9D,F5 ;24061          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;24062                                           ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1F3E, 3210,15 ;24063          WRITE.MEM LS[T8]                 ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;24064                                           ; ADDR 0 TO PHYSICAL ADDR 0, SET BYTE
                ;24065                                           ; OFFSET BIT
                ;24066
U 1F3F, FC88,15 ;24067          DEC LS[ADDRESS.DATA]             ;OTHER DATA IS 3 (ADDRESS 3)
U 1F40, 3642,15 ;24068          MOV LS[#2] TO WR[0]              ;2 IN WR
U 1F41, 3EB0,15 ;24069          MOV WR[0] TO LS[T58]             ;DATA OF 2 TO WRITE IN UBE ADDRESS REG
U 1F42, B626,15 ;24070          MOV LS[#FF] TO WR[0]             ;000000FF IN WR
U 1F43, BE1E,15 ;24071          MOV WR[0] TO LS[T15]             ;EXPECTED DATA
U 1F44, 09F4,6C ;24072          JSR [LOOP.T3D.2]                 ;PERFORM TEST WITH BYTE OFFSET SET TO
                ;24073                                           ; ADDRESS 2
U 1F45, 89F5,84 ;24074          JMP [END.T3D]                    ;DONE
                ;24075
                ;24076  ;
                ;24077  ;SUBROUTINE TO PERFORM NPR TEST
                ;24078  ;
                ;24079  LOOP.T3D.2:
U 1F46, B640,15 ;24080          MOV LS[#1] TO WR[0]              ;1 IN WR
U 1F47, BE82,15 ;24081          MOV WR[0] TO LS[ERROR.NUMBER]    ;SET UP ERROR NUMBER IN LS
                ;24082
U 1F48, 19B8,35 ;24083          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                ;24084                                           ;SET UP TO WRITE TO UBE ADDRESS REG
U 1F49, 32B0,15 ;24085          WRITE.MEM LS[T58]                ;PUT A 2 OR 4 IN ADDRESS REG
U 1F4A, 99B6,35 ;24086          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                ;24087                                           ;SET UP TO WRITE TO UBE CYCLE COUNT
U 1F4B, 329E,15 ;24088          WRITE.MEM LS[ONES]               ;PUT A -1 IN CYCLE COUNT (DO 1 CYCLE)
U 1F4C, 99BA,35 ;24089          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                ;24090                                           ;SET UP TO WRITE TO UBE CONTROL REG 1
U 1F4D, 321C,15 ;24091          WRITE.MEM LS[T14]                ;SET UP CONTROL REG TO ISSUE NPR TO
                ;24092                                           ; READ ARRAY AND INTR AT BR6 ON DONE
                ;24093  LOOP.T3D.INT:
U 1F4E, 09F4,E7 ;24094          JMP.IF[NO.INTERRUPT] TO [LOOP.T3D.INT]
                ;24095                                           ;WAIT FOR BR INTERRUPT
                ;24096
U 1F4F, 8A1C,DC ;24097          JSR [CHECK.UBE.ERR]              ;SEE IF UBE ERR AND ISSUE NECESSARY
                ;24098                                           ; GRANTS
U 1F50, 89F4,64 ;24099          JMP [LOOP.T3D.2]                 ;ERROR LOOP RETURN IF ENABLED
                ;24100
```

```
U 1F51, FF82,15 ;24101          INC LS[ERROR.NUMBER]              ;ERROR 2
U 1F52, 98B5,B5 ;24102          MEM.REQ[READ.P] ADRS[BEDB] DT[WORD]
                ;24103                                            ;SET UP TO READ DATA IN UBE
U 1F53, 3022,15 ;24104          MOV MEM.DATA TO WR[0]             ;GET RESULT
U 1F54, B61E,95 ;24105          MOV LS[T15] TO WR[1]              ;EXPECTED DATA
U 1F55, 0869,3C ;24106          JSR [CHECK.RESULT]                ;SEE IF ARRAY WRITTEN CORRECTLY
U 1F56, 89F4,64 ;24107          JMP [LOOP.T3D.2]                  ;ERROR LOOP IF ENABLED
                ;24108
U 1F57, 5B00,14 ;24109          RETURN                            ;DONE
                ;24110  END.T3D:
```

```
;24111  .page   "  TEST 3E UNIBUS DATI With CRD and RDS Error Test (M8391 MCT Module)"
;24112  ;++
;24113  ;
;24114  ; Test Description:
;24115  ;
;24116  ;       This test checks the micro-code associated with DATI reads from the
;24117  ;       the array when single or double bit errors are detected.
;24118  ;         The test writes single bit errors in memory and attempts to read
;24119  ;       that location.  The micro-code should branch to the routine to cor-
;24120  ;       rect the single bit error before sending the data back to the UBE.
;24121  ;         After this has been verified, double bits errors are written into
;24122  ;       memory and the flow which aborts the UNIBUS read to array is checked.
;24123  ;       The test also checks the RDS bit in UB CSR2 to assure that it sets
;24124  ;       when a double error occurs, but not when a single bit error occurs.
;24125  ;       The test is done twice, with a double error in the second cycle only
;24126  ;       and in the first cycle only, to check both double error micro code
;24127  ;       paths.
;24128  ;
;24129  ; Assumptions:
;24130  ;
;24131  ;       It is assumed that all other micro-diagnostics have run successfully.
;24132  ;       It is also assumed that the UBE present is known to be good.
;24133  ;
;24134  ; Test Steps:
;24135  ;
;24136  ;       1)  Set up error mask, error number, and module number in LS (for
;24137  ;           error printouts) and clear previous error number in LS.  Also
;24138  ;           issue DCLO to UBE to clear error bits.
;24139  ;       2)  Set up UNIBUS MAP to map virtual address 0 to physical address 0
;24140  ;           and set the VALID and BYTE OFFSET bits.
;24141  ;       3)  Clear the IPL bits in the PSW (allow all interrupts).  Write a
;24142  ;           1000000 at physical memory address 0 and a 1 at address 4 with
;24143  ;           CKBTS for data of all 0's (single bit error in byte 3 of longword
;24144  ;           0 and byte 0 of longword at address 4).
;24145  ;       4)  Set up UBE data reg with all 1's initially, set up UBE cycle count
;24146  ;           to 1 cycle (FFFF), and address reg to address 2.  Clear UB CSR2
;24147  ;           error bits by writing to UB CSR2.
;24148  ;       5)  Set up control reg 1 of the UBE to execute one DATI cycle via an
;24149  ;           NPR and interrupt via a BR6 when done.  Also set the GO bit to
;24150  ;           start the UBE.
;24151  ;       6)  Loop waiting for BR interrupt from UBE signifying that UBE is done.
;24152  ;       7)  Check the identifier code when BR interrupt occurs.  If a BR 7 was
;24153  ;           received (indicates a UBE error), read the UBE error register and
;24154  ;           report the error.
;24155  ;       8)  Read UBE data reg and check for data of all 0's.
;24156  ;       9)  Read UB CSR2 (on MCT) and check for zeros in error bits.  Read CSR
;24157  ;           1 and check for zeros in error bits.
;24158  ;       10) Write address 0 with data of 0 and normal CKBTs.  Write address 4
;24159  ;           with data of 3(H) and CKBTS for data of all zeros.  Repeat steps 4
;24160  ;           through 6 above.
;24161  ;       11) Read UBE control reg 2 (error reg) and check that NO SSYN error bit
;24162  ;           is set.
;24163  ;       12) Read UB CSR2 (on MCT) and check for UB RDS (bit 31) set, other clear.
;24164  ;           Read CSR 1 and check for all error bits clear.
;24165  ;       13) Write address 0 with data of 3 and CKBTs for data of all 0's.
```

```
;24166  ;          Write address 4 with data of all 0's and normal CKBTS. Repeat steps
;24167  ;          4 through 6, 11 and 12 above.
;24168  ;
;24169  ; Errors:
;24170  ;
;24171  ;       Error 1 - BR7 interrupt occurred when doing DATO (indicates UBE error).
;24172  ;                 Received data is contents of UBE control reg 2 (error reg-
;24173  ;                 ister).
;24174  ;       Error 2 - UBE did not receive corrected memory data on DATI with single
;24175  ;                 bit error in each longword on 2 MEM CYC read.
;24176  ;       Error 3 - UB CSR2 had one or more error bits set on correctable error.
;24177  ;       Error 4 - CSR 1 had one or more error bits set after UNIBUS CRD error.
;24178  ;       Error 5 - UBE control reg 2 (error reg) did not have NO SSYN error set
;24179  ;                 on DATI to location with double error in second cycle of
;24180  ;                 array access.
;24181  ;       Error 6 - UB CSR2 did not have RDS bit set or other bits set on DATI
;24182  ;                 to location with double error in second cycle of array access.
;24183  ;       Error 7 - CSR 1 had one or more error bits set after UNIBUS RDS error.
;24184  ;       Error 8 - UBE control reg 2 (error reg) did not have NO SSYN error set
;24185  ;                 on DATI to location with double error in first cycle of array
;24186  ;                 access.
;24187  ;       Error 9 - UB CSR2 did not have RDS bit set or other bits set on DATI
;24188  ;                 to location with double error in first cycle of array access.
;24189  ;       Error A - CSR 1 had one or more error bits set after UNIBUS RDS error.
;24190  ;       Error B - UBE control reg 2 (error reg) did not have NO SSYN error set
;24191  ;                 on DATI to array when UB L RDS had not been cleared.
;24192  ;       Error C - UB CSR2 did not have RDS bit set or other bits set on DATI
;24193  ;                 to array when UB L RDS had not been cleared.
;24194  ;       Error D - CSR 1 had one or more error bits set after UNIBUS RDS error.
;24195  ;       Error E - BR7 interrupt occurred when doing DATO (indicates UBE error).
;24196  ;                 Received data is contents of UBE control reg 2 (error reg-
;24197  ;                 ister).
;24198  ;       Error F - UBE did not receive correct memory data on DATI with good
;24199  ;                 data in array.
;24200  ;       Error 10- UB CSR2 had one or more error bits set on access to good data
;24201  ;                 (UB L RDS failed to be cleared).
;24202  ;       Error 11- CSR 1 had one or more error bits set after access to good
;24203  ;                 array location.
;24204  ;
;24205  ; Debug:
;24206  ;
;24207  ;       Error 1 - This error indicates that the UBE detected an error.  See the
;24208  ;       UBE error register list in this listing to determine what went wrong.
;24209  ;
;24210  ;       Error 2 - This error most likely indicates a problem with the micro-
;24211  ;       code.  The micro-flow should take the ERR path at UB CYC TWO ARY CYC
;24212  ;       C7 and the SERR L path at UB CYC TWO ARY CYC RD WDE C8 during the first
;24213  ;       memory cycle, and the ERR path at UB CYC RD C7 and the SERR path at UB
;24214  ;       CYC ERR C8 during the second memory cycle.
;24215  ;
;24216  ;       Error 3 - This error most likely indicates a problem with the CSR CON-
;24217  ;       TROL PAL.  Check the signal UB L RDS L for a high after the cycle that
;24218  ;       asserts CSR ERR ADR CLK.  If incorrect, check the input CPH/UBL for a
;24219  ;       low when CSR ERR ADR CLK is asserted.  If this is correct, suspect the
;24220  ;       CSR CONTROL PAL itself.
```

```
                ;24221  ;
                ;24222  ;       Error 4 - If the CRD bit (bit 30) is asserted, this error indicates a
                ;24223  ;       problem with the CSR CONTROL PAL or the input CPH/UBL.  Check the
                ;24224  ;       signal L CRD H for a low after the the cycle that asserts CSR ERR ADR
                ;24225  ;       CLK.  If incorrect, check the signal CPH/UBL for a low at the same
                ;24226  ;       time.  If CPH/UBL is OK, then the PAL is probably bad.
                ;24227  ;         If other error bits are set, suspect the PAL used to produce that
                ;24228  ;       bit.
                ;24229  ;
                ;24230  ;       Error 5 - This error most likely indicates a problem with the micro
                ;24231  ;       code.  The branch on SERR not at UB CYC ERR C8 is most suspect.  An-
                ;24232  ;       other possibility is the STALL MBSY did not block the issuing of
                ;24233  ;       SSYN by the POWER UP AND INIT PAL.  Check the signal ENABLE ERROR H
                ;24234  ;       for a high during UB CYC RD C7.  Trace it back if it does not go high.
                ;24235  ;       If ENABLE ERROR H is OK, then L ISSYN should be prevented from becoming
                ;24236  ;       asserted high.  If not, the POWER UP AND INIT PAL is probably bad.
                ;24237  ;
                ;24238  ;       Error 6 - This error indicates a problem with the CSR CONTROL PAL or
                ;24239  ;       the UB CSR2 PAL.  Check the signal UB L RDS L out of the CSR CONTROL
                ;24240  ;       PAL for a low after the cycle that asserts CSR ERR ADR CLK.  If this
                ;24241  ;       is incorrect, the PAL is probably bad.
                ;24242  ;         If UB L RDS L is low, check it at the input to the UB CSR2 PAL.  If
                ;24243  ;       OK there, the UB CSR2 PAL is probably bad.
                ;24244  ;
                ;24245  ;       Error 7 - Same as error 4.
                ;24246  ;
                ;24247  ;       Error 8 - Same as error 5 except the branch is at UB CYC TWO ARY CYC
                ;24248  ;       RD WDE C8.
                ;24249  ;
                ;24250  ;       Error 9 - Same as error 6
                ;24251  ;
                ;24252  ;       Error A - Same as error 4.
                ;24253  ;
                ;24254  ;       Error B - See error C below.
                ;24255  ;
                ;24256  ;       Error C - This error, and error B above, indicates that UB L RDS L has
                ;24257  ;       been cleared out even though no write to CSR 2 (which should clear the
                ;24258  ;       error bits) has been performed.  Check the signal CLR UBRDS L to make
                ;24259  ;       sure that it is not stuck low to the input of the CSR DATA ERROR
                ;24260  ;       CONTROL PAL.  If this is OK, suspect the DATA ERROR CONTROL PAL itself.
                ;24261  ;
                ;24262  ;       Error D - Same as error 4.
                ;24263  ;
                ;24264  ;       Error E - This error indicates that UB L RDS L from the CSR DATA
                ;24265  ;       CONTROL PAL or BUS MC D31 from the UB CSR 2 PAL is not getting cleared
                ;24266  ;       by a write to CSR 2, thus ????
                ;24267  ;
                ;24268  ;--
                ;24269
                ;24270  ;       T14 = DATA FOR UBE CONTROL REG 1
                ;24271  ;       T15 = EXPECTED DATA IN UB ERROR REG (NO SSYN AND CCOVF SET)
                ;24272  ;       T58 = ADDRESS 5 FOR WRITING ERRORS IN LONGWORD AT ADDRESS 4
                ;24273
                ;24274  T.3E:
U 1F58, B65E,15 ;24275          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
```

```
                    ;24276                                          ; STATUS WORD
U 1F59, 3E80,15     ;24277          MOV WR[0] TO LS[CONTROL.STATUS]  ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;24278                                          ; BIT 15 INDICATES START OF TEST TO
                    ;24279                                          ; CONSOLE
U 1F5A, 10E0,15     ;24280          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;24281  WAIT.T3E.0:
U 1F5B, 89F5,B4     ;24282          JMP [WAIT.T3E.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;24283                                          ; RESPOND
                    ;24284
U 1F5C, 0A1B,8C     ;24285          JSR [ISSUE.DCLO]                ;INIT UBE
U 1F5D, 65A0,15     ;24286          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 1F5E, 3644,15     ;24287          MOV LS[MCT] TO WR[0]            ;SET BIT 2 IN WR
U 1F5F, 3E8C,15     ;24288          MOV WR[0] TO LS[MODULE.NUM]      ;SET UP TO CALL OUT MCT MODULE IF ERROR
U 1F60, 367E,15     ;24289          MOV LS[BIT31] TO WR[0]          ;SET BIT 31 IN WR
U 1F61, 4772,15     ;24290          BIS LS[BIT25] TO WR[0]          ;SET BIT 25 IN WR (BYTE OFFSET)
U 1F62, 3E10,15     ;24291          MOV WR[0] TO LS[T8]              ;STORE IN LS
U 1F63, 1B9D,F5     ;24292          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                    ;24293                                          ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 1F64, 3210,15     ;24294          WRITE.MEM LS[T8]                ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                    ;24295                                          ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                    ;24296                                          ; OFFSET SET
U 1F65, 2F80,15     ;24297          CLR WR[0]                       ;CLEAR WR 0
U 1F66, BFFE,15     ;24298          MOV WR[0] TO LS[PSL.HW]          ;CLEAR IPL BITS IN PSW
U 1F67, B700,15     ;24299          MOV LS[CKBTS.0111100] TO WR[0]  ;GET CKBTS FOR DATA OF 0 IN WR 0
U 1F68, A0C0,15     ;24300          COM WR[0]                       ;COMPLEMENT CKBTS TO CORRECT FOR HARDWARE
U 1F69, 452C,15     ;24301          BIC LS[#FFFFFF00] TO WR[0]      ;CLEAR ALL BUT LOW BYTE
U 1F6A, 8A17,FC     ;24302          JSR [WRITE.CSR1]                ;PUT CKBTS FOR DATA OF 0 IN CSR 1, CLEAR
                    ;24303                                          ; ALL OTHER BITS IN CSR
U 1F6B, 3644,15     ;24304          MOV LS[#4] TO WR[0]             ;4 IN WR
U 1F6C, 2040,15     ;24305          INC WR[0]                       ;5 IN WR
U 1F6D, 3EB0,15     ;24306          MOV WR[0] TO LS[T58]             ;5 IN LS
U 1F6E, 8A1C,7C     ;24307          JSR [SETUP.DATI.NPR]            ;SETUP T14 WITH UBE DATA TO ISSUE NPR
                    ;24308                                          ; DATI, INTERRUPT ON DONE AT BR6
                    ;24309
U 1F6F, 3641,15     ;24310          MOV LS[#1] TO WR[2]             ;STARTING ERROR NUMBER (ERROR 1)
U 1F70, 19B4,35     ;24311          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                    ;24312                                          ;SET UP TO WRITE TO UBE DATA REG
U 1F71, 329E,15     ;24313          WRITE.MEM LS[ONES]              ;PUT 1'S IN DATA REG TO INITIALIZE
U 1F72, 1D40,F5     ;24314          MEM.REQ[MAINT.ECC.DATA] ADRS[#1] DT[LONG]
                    ;24315                                          ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                    ;24316                                          ; SET FOR BRANCH
U 1F73, 3270,15     ;24317          WRITE.MEM LS[#1000000]          ;WRITE DATA OF 1000000 AT LOCATION 0
                    ;24318                                          ; WITH CKBTS FOR DATA OF ALL 0'S
U 1F74, 1DB0,F5     ;24319          MEM.REQ[MAINT.ECC.DATA] ADRS[T58] DT[LONG]
                    ;24320                                          ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                    ;24321                                          ; SET FOR BRANCH
U 1F75, 3240,15     ;24322          WRITE.MEM LS[#1]                ;WRITE DATA OF 1 AT LOCATION 4 WITH
                    ;24323                                          ; CKBTS FOR DATA OF ALL 0'S
U 1F76, 8A1E,3C     ;24324          JSR [CLEAR.CSR2]                ;CLEAR ERROR BITS IN UB CSR2
U 1F77, 89F9,5C     ;24325          JSR [LOOP.T3E.3.4]              ;PERFORM TEST WITH CRD ERRORS AT BOTH
                    ;24326                                          ; LOCATIONS
                    ;24327
U 1F78, 4045,15     ;24328          ADD LS[#4] TO WR[2]              ;STARTING ERROR NUMBER (ERROR 5)
U 1F79, 999C,75     ;24329          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                    ;24330                                          ;SET UP TO WRITE AT ADDRESS 0
```

```
U 1F7A, B29C,15 ;24331          WRITE.MEM LS[ZERO]                ;WRITE DATA OF 0'S WITH NORMAL CKBTS
U 1F7B, 1DB0,F5 ;24332          MEM.REQ[MAINT.ECC.DATA] ADRS[T58] DT[LONG]
                ;24333                                            ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;24334                                            ; SET FOR BRANCH
U 1F7C, B2C0,15 ;24335          WRITE.MEM LS[#3(H)]               ;WRITE DATA OF 3 AT LOCATION 4 WITH
                ;24336                                            ; CKBTS FOR DATA OF ALL 0'S
U 1F7D, 3650,15 ;24337          MOV LS[NO.SSYN.ERR] TO WR[0]      ;EXPECTED BIT IN UBE ERROR REG
U 1F7E, 475A,15 ;24338          BIS LS[CCOVF] TO WR[0]            ;ALSO CCOVF EXPECTED TO BE SET
U 1F7F, BE1E,15 ;24339          MOV WR[0] TO LS[T15]              ;STORE EXPECTED DATA IN LS
U 1F80, 8A1E,3C ;24340          JSR [CLEAR.CSR2]                  ;CLEAR ERROR BITS IN UB CSR2
U 1F81, 09FB,AC ;24341          JSR [LOOP.T3E.6.7]                ;PERFORM TEST WITH RDS ERROR IN SECOND
                ;24342                                            ; CYCLE
                ;24343
U 1F82, 40C1,15 ;24344          ADD LS[#3(H)] TO WR[2]            ;STARTING ERROR NUMBER (ERROR 8)
U 1F83, 1D40,F5 ;24345          MEM.REQ[MAINT.ECC.DATA] ADRS[#1] DT[LONG]
                ;24346                                            ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;24347                                            ; SET FOR BRANCH
U 1F84, B2C0,15 ;24348          WRITE.MEM LS[#3(H)]               ;WRITE DATA OF 3 AT LOCATION 0 WITH
                ;24349                                            ; CKBTS FOR DATA OF ALL 0'S
U 1F85, 9944,75 ;24350          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                ;24351                                            ;SET UP TO WRITE AT ADDRESS 4
U 1F86, B29C,15 ;24352          WRITE.MEM LS[ZERO]                ;WRITE DATA OF 0'S WITH NORMAL CKBTS
U 1F87, 8A1E,3C ;24353          JSR [CLEAR.CSR2]                  ;CLEAR ERROR BITS IN UB CSR2
U 1F88, 09FB,AC ;24354          JSR [LOOP.T3E.6.7]                ;PERFORM TEST WITH RDS ERROR IN FIRST
                ;24355                                            ; CYCLE
                ;24356
U 1F89, 40C1,15 ;24357          ADD LS[#3(H)] TO WR[2]            ;STARTING ERROR NUMBER (ERROR B)
U 1F8A, 1D40,F5 ;24358          MEM.REQ[MAINT.ECC.DATA] ADRS[#1] DT[LONG]
                ;24359                                            ;SET UP TO USE DIAGNOSTIC ROUTINE  LVA00
                ;24360                                            ; SET FOR BRANCH
U 1F8B, B29C,15 ;24361          WRITE.MEM LS[ZERO]                ;WRITE DATA OF 0'S AT LOCATION 0 WITH
                ;24362                                            ; CKBTS FOR DATA OF ALL 0'S
U 1F8C, 9944,75 ;24363          MEM.REQ[WRITE.P] ADRS[#4] DT[LONG]
                ;24364                                            ;SET UP TO WRITE AT ADDRESS 4
U 1F8D, B29C,15 ;24365          WRITE.MEM LS[ZERO]                ;WRITE DATA OF 0'S WITH NORMAL CKBTS
U 1F8E, 365A,15 ;24366          MOV LS[CCOVF] TO WR[0]            ;CCOVF EXPECTED TO BE SET IN UB CSR 2
U 1F8F, BE1E,15 ;24367          MOV WR[0] TO LS[T15]              ;STORE EXPECTED DATA IN LS
                ;24368
U 1F90, 09FB,AC ;24369          JSR [LOOP.T3E.6.7]                ;PERFORM TEST WITH NO ERRORS IN DATA
                ;24370                                            ; BUT WITHOUT CLEARING UB L RDS L
                ;24371
U 1F91, 40C1,15 ;24372          ADD LS[#3(H)] TO WR[2]            ;STARTING ERROR NUMBER (ERROR E)
U 1F92, 8A1E,3C ;24373          JSR [CLEAR.CSR2]                  ;CLEAR ERROR BITS IN UB CSR2
U 1F93, 89F9,5C ;24374          JSR [LOOP.T3E.3.4]                ;PERFORM TEST WITH GOOD DATA AFTER
                ;24375                                            ; CLEARING UB L RDS L WITH WRITE TO
                ;24376                                            ; CSR 2
                ;24377
U 1F94, 09FE,14 ;24378          JMP [END.T3E]                     ;DONE
                ;24379
                ;24380  ;
                ;24381  ;SUBROUTINES TO PERFORM NPR TEST
                ;24382  ;
                ;24383
                ;24384  LOOP.T3E.3.4:
U 1F95, 5F28,15 ;24385          MCOM LS[#FFFF] TO WR[0]           ;CLEAR BITS 15-0
```

```
U 1F96, 3E8A,15 ;24386          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK BITS 15-0
                ;24387  LOOP.T3E.2:
U 1F97, 3E83,15 ;24388          MOV WR[2] TO LS[ERROR.NUMBER]   ;SET UP ERR NUMBER IN LS (ERROR 1 OR E)
                ;24389  LOOP.T3E.1:
U 1F98, 19B8,35 ;24390          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                ;24391                                          ;SET UP TO WRITE TO UBE ADDRESS REG
U 1F99, B242,15 ;24392          WRITE.MEM LS[#2]                ;PUT ADDRESS 2 IN UBE ADDRESS REG
U 1F9A, 99B6,35 ;24393          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                ;24394                                          ;SET UP TO WRITE TO UBE CYCLE COUNT
U 1F9B, 329E,15 ;24395          WRITE.MEM LS[ONES]              ;PUT A -1 IN CYCLE COUNT (DO 1 CYCLE)
U 1F9C, 99BA,35 ;24396          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                ;24397                                          ;SET UP TO WRITE TO UBE CONTROL REG 1
U 1F9D, 321C,15 ;24398          WRITE.MEM LS[T14]               ;SET UP CONTROL REG TO ISSUE NPR TO
                ;24399                                          ; WRITE A WORD TO ARRAY AND INTR AT
                ;24400                                          ; BR6 ON DONE
                ;24401  LOOP.T3E.INT.1:
U 1F9E, 89F9,E7 ;24402          JMP.IF[NO.INTERRUPT] TO [LOOP.T3E.INT.1] ;LOOP UNTIL BR INTERRUPT
                ;24403
U 1F9F, 8A1C,DC ;24404          JSR [CHECK.UBE.ERR]             ;SEE IF UBE ERR AND ISSUE NECESSARY
                ;24405                                          ; GRANTS
U 1FA0, 89F9,84 ;24406          JMP [LOOP.T3E.1]                ;ERROR LOOP RETURN IF ENABLED
                ;24407
U 1FA1, FF82,15 ;24408          INC LS[ERROR.NUMBER]            ;ERROR 2 OR F
U 1FA2, 98B5,B5 ;24409          MEM.REQ[READ.P] ADRS[BEDB] DT[WORD]
                ;24410                                          ;SET UP TO READ DATA IN UBE DATA BUFFER
U 1FA3, 3022,15 ;24411          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1FA4, 2F82,95 ;24412          CLR WR[1]                       ;EXPECTED DATA
U 1FA5, 0869,3C ;24413          JSR [CHECK.RESULT]              ;SEE IF ARRAY READ AND CORRECTED COR-
                ;24414                                          ; RECTLY
U 1FA6, 89F9,74 ;24415          JMP [LOOP.T3E.2]                ;ERROR LOOP IF ENABLED
                ;24416
U 1FA7, FF82,15 ;24417          INC LS[ERROR.NUMBER]            ;ERROR 3 OR 10
U 1FA8, 1D47,75 ;24418          MEM.REQ[READ.CSR] ADRS[CSR2] DT[LONG]
                ;24419                                          ;SET UP TO READ DATA IN UB CSR2
U 1FA9, 3022,15 ;24420          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1FAA, B630,95 ;24421          MOV LS[CSR2.MASK] TO WR[1]      ;SET UP ERROR MASK FOR UB CSR2
U 1FAB, BE8A,95 ;24422          MOV WR[1] TO LS[ERROR.MASK]     ;STORE IN LS
U 1FAC, 2F82,95 ;24423          CLR WR[1]                       ;EXPECTED DATA (ERROR BITS CLEAR)
U 1FAD, 0869,3C ;24424          JSR [CHECK.RESULT]              ;SEE IF UB CSR2 IS CORRECT
U 1FAE, 09F9,54 ;24425          JMP [LOOP.T3E.3.4]              ;ERROR LOOP IF ENABLED
                ;24426
U 1FAF, FF82,15 ;24427          INC LS[ERROR.NUMBER]            ;ERROR 4 OR 11
U 1FB0, 362E,15 ;24428          MOV LS[CSR1.MASK] TO WR[0]      ;ERROR MASK FOR CSR 1
U 1FB1, 3E8A,15 ;24429          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK ALL ERROR & CONTROL BITS IN CSR1
U 1FB2, 9D45,75 ;24430          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;24431                                          ;SET UP TO READ CSR1
U 1FB3, 3022,15 ;24432          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1FB4, 2F82,95 ;24433          CLR WR[1]                       ;EXPECTED RESULT
U 1FB5, 0869,3C ;24434          JSR [CHECK.RESULT]              ;CHECK THAT ALL ERROR BITS ARE CLEAR
U 1FB6, 09F9,54 ;24435          JMP [LOOP.T3E.3.4]              ;ERROR LOOP IF ENABLED
                ;24436
U 1FB7, 99BC,35 ;24437          MEM.REQ[WRITE.P] ADRS[CLEAR.ERR.ADDR] DT[WORD]
                ;24438                                          ;SET UP TO CLEAR UBE ERROR BITS IF SET
U 1FB8, B29C,15 ;24439          WRITE.MEM LS[ZERO]              ;CLEAR UBE ERROR REG
U 1FB9, 5B00,14 ;24440          RETURN                          ;DONE WITH NPR TEST FOR THIS DATA
```

```
                 :24441
                 :24442
                 :24443  LOOP.T3E.6.7:
U 1FBA, 3E83,15  :24444          MOV WR[2] TO LS[ERROR.NUMBER]   ;SET UP ERR NUMBER IN LS (ERROR 5, 8
                 :24445                                          ; OR B)
U 1FBB, 5F28,15  :24446          MCOM LS[#FFFF] TO WR[0]         ;CLEAR BITS 15-0
U 1FBC, 3E8A,15  :24447          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK BITS 15-0
                 :24448  LOOP.T3E.5:
U 1FBD, 19B8,35  :24449          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                 :24450                                          ;SET UP TO WRITE TO UBE ADDRESS REG
U 1FBE, B242,15  :24451          WRITE.MEM LS[#2]                ;PUT ADDRESS 2 IN UBE ADDRESS REG
U 1FBF, 99B6,35  :24452          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                 :24453                                          ;SET UP TO WRITE TO UBE CYCLE COUNT
U 1FC0, 329E,15  :24454          WRITE.MEM LS[ONES]              ;PUT A -1 IN CYCLE COUNT (DO 1 CYCLE)
U 1FC1, 99BA,35  :24455          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                 :24456                                          ;SET UP TO WRITE TO UBE CONTROL REG 1
U 1FC2, 321C,15  :24457          WRITE.MEM LS[T14]               ;SET UP CONTROL REG TO ISSUE NPR TO
                 :24458                                          ; READ A WORD FROM ARRAY AND INTR AT
                 :24459                                          ; BR6 ON DONE
                 :24460  LOOP.T3E.INT.5:
U 1FC3, 09FC,37  :24461          JMP.IF[NO.INTERRUPT] TO [LOOP.T3E.INT.5] ;LOOP UNTIL BR INTERRUPT
                 :24462
U 1FC4, 9F46,35  :24463          MEM.REQ[ISSUE.BG] ADRS[BG6] DT[WORD]
                 :24464                                          ;ISSUE BG ON LEVEL 6
U 1FC5, 3022,15  :24465          MOV MEM.DATA TO WR[0]           ;COMPLETE BG CYCLE
U 1FC6, DB00,15  :24466          NOP                             ;ALLOW TIME FOR BR7 INTR
U 1FC7, 9FCE,35  :24467          MEM.REQ[ISSUE.BG] ADRS[BG7] DT[WORD]
                 :24468                                          ;ISSUE BG ON LEVEL 7
U 1FC8, 3022,15  :24469          MOV MEM.DATA TO WR[0]           ;COMPLETE BG CYCLE
                 :24470
U 1FC9, 98BF,B5  :24471          MEM.REQ[READ.P] ADRS[BECR2] DT[WORD]
                 :24472                                          ;SET UP TO READ DATA UBE ERROR REG
U 1FCA, 3022,15  :24473          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1FCB, B61E,95  :24474          MOV LS[T15] TO WR[1]            ;EXPECTED DATA (NO SSYN ERR AND CCOVF
                 :24475                                          ; FOR ERRORS 5 AND 8, CCOVF ONLY FOR
                 :24476                                          ; ERROR B)
U 1FCC, 0869,3C  :24477          JSR [CHECK.RESULT]              ;SEE IF NO SSYN ERROR SET
U 1FCD, 09FB,D4  :24478          JMP [LOOP.T3E.5]                ;ERROR LOOP IF ENABLED
                 :24479
U 1FCE, FF82,15  :24480          INC LS[ERROR.NUMBER]            ;ERROR 6, 9 OR C
U 1FCF, 1D47,75  :24481          MEM.REQ[READ.CSR] ADRS[CSR2] DT[LONG]
                 :24482                                          ;SET UP TO READ UB CSR2
U 1FD0, 3022,15  :24483          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 1FD1, B630,95  :24484          MOV LS[CSR2.MASK] TO WR[1]      ;SET UP ERROR MASK FOR UB CSR2
U 1FD2, BE8A,95  :24485          MOV WR[1] TO LS[ERROR.MASK]     ;STORE IN LS
U 1FD3, B67E,95  :24486          MOV LS[BIT31] TO WR[1]          ;EXPECTED DATA (UB RDS BIT SET)
U 1FD4, 0869,3C  :24487          JSR [CHECK.RESULT]              ;SEE IF UB CSR2 IS CORRECT
U 1FD5, 89FB,A4  :24488          JMP [LOOP.T3E.6.7]              ;ERROR LOOP IF ENABLED
                 :24489
U 1FD6, FF82,15  :24490          INC LS[ERROR.NUMBER]            ;ERROR 7, A OR D
U 1FD7, 362E,15  :24491          MOV LS[CSR1.MASK] TO WR[0]      ;ERROR MASK FOR CSR 1
U 1FD8, 3E8A,15  :24492          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK ALL ERROR & CONTROL BITS IN CSR1
U 1FD9, 9D45,75  :24493          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                 :24494                                          ;SET UP TO READ CSR1
U 1FDA, 3022,15  :24495          MOV MEM.DATA TO WR[0]           ;GET RESULT
```

```
U 1FDB, 2F82,95 ;24496          CLR WR[1]                           ;EXPECTED RESULT
U 1FDC, 0869,3C ;24497          JSR [CHECK.RESULT]                  ;CHECK THAT ALL ERROR BITS ARE CLEAR
U 1FDD, 89FB,A4 ;24498          JMP [LOOP.T3E.6.7]                  ;ERROR LOOP IF ENABLED
                ;24499
U 1FDE, 99BC,35 ;24500          MEM.REQ[WRITE.P] ADRS[CLEAR.ERR.ADDR] DT[WORD]
                ;24501                                              ;SET UP TO CLEAR UBE ERROR BITS IF SET
U 1FDF, B29C,15 ;24502          WRITE.MEM LS[ZERO]                  ;CLEAR UBE ERROR REG
U 1FE0, 5B00,14 ;24503          RETURN                              ;DONE WITH NPR TEST FOR THIS DATA
                ;24504
                ;24505  END.T3E:
```

```
;24506  .page   "  TEST 3F UB CSR 2 and UB ERRSUM On DATI Test  (M8391 MCT Module)"
;24507  ;++
;24508  ;
;24509  ; Test Description:
;24510  ;
;24511  ;       This test checks the branch paths on UB ERRSUM in the UNIBUS READ TO
;24512  ;       ARRAY micro code.
;24513  ;         The test first sets the TB PAR DIAG bit in CSR 1 and writes UBS MAP
;24514  ;       location 0.  The map thus contains a parity error.  UBS MAP location
;24515  ;       1 is written without a parity error.  The UBE tries to read the array
;24516  ;       at address 1FE with byte offset both clear and then set.  This checks
;24517  ;       the detection of UB ERRSUM at UB CYC RD C6 and UB CYC TWO ARY CYC RD
;24518  ;       C6.  The test will check that in both cases, the NO SSYN ERR bit in UBE
;24519  ;       CONTROL REG 2 is set, and that the UB TB PAR ERR bit in UB CSR 2 (on
;24520  ;       the MCT module) is set.  Then the parity error is cleared in UB MAP
;24521  ;       location 0, a parity error is written in UB MAP location 1, and the
;24522  ;       test repeated with byte offset set.  This checks the UB ERRSUM branch
;24523  ;       at the cycle which executes after the 2ND CYC branch from UB CYC RD C5.
;24524  ;
;24525  ; Assumptions:
;24526  ;
;24527  ;       It is assumed that all other micro-diagnostics have run successfully.
;24528  ;       It is also assumed that the UBE present is known to be good.
;24529  ;
;24530  ; Test Steps:
;24531  ;
;24532  ;       1)  Set up error mask, error number, and module number in LS (for
;24533  ;           error printouts) and clear previous error number in LS.  Also
;24534  ;           issue DCLO to UBE to clear error bits.
;24535  ;       2)  Set up UNIBUS MAP to map virtual address 0 to physical address 0
;24536  ;           and set the VALID bit.  Write this location with bad parity.  Set
;24537  ;           up address 1 to map virtual 200 to physical 200 and set VALID bit.
;24538  ;       3)  Clear the IPL bits in the PSW (allow all interrupts).  Write all
;24539  ;           zeros at address 1FC and 200.
;24540  ;       4)  Set up UBE cycle count to 1 cycle (FFFF) and address reg to address
;24541  ;           1FE.
;24542  ;       5)  Set up control reg 1 of the UBE to execute one DATI cycle via an
;24543  ;           NPR and interrupt via a BR6 when done.  Also set the GO bit to
;24544  ;           start the UBE.
;24545  ;       6)  Loop waiting for BR interrupt from UBE signifying that UBE is done.
;24546  ;       7)  Issue bus grant on level 6, then level 7 to clear BRs.
;24547  ;       8)  Read UBE control reg 2 (error reg).  Check that NO SSYN ERR bit is
;24548  ;           set.
;24549  ;       9)  Read UB CSR 2 and check that UB TB PAR ERR is set.
;24550  ;       10) Repeat steps 4 through 9 above with UB MAP set up to execute 2 MEM
;24551  ;           CYCLES (BYTE OFFSET set).
;24552  ;       11) Repeat steps 4 through 9 with TB PAR ERROR in second map location.
;24553  ;
;24554  ; Errors:
;24555  ;
;24556  ;       Error 1 - UB CONTROL REG 2 did not have NO SSYN ERR set after DATI to
;24557  ;                 array with a UB TB ERR and 1 MEM CYC.
;24558  ;       Error 2 - UB TB PAR ERR bit in UB CSR 2 (on MCT module) not set after
;24559  ;                 DATI to array with UB TB PAR ERR and 1 MEM CYC.
;24560  ;       Error 3 - UB CONTROL REG 2 did not have NO SSYN ERR set after DATI to
```

```
                    ;24561  ;                   array with a UB TB PAR ERR in first cycle of two cycle read.
                    ;24562  ;       Error 4 - UB TB PAR ERR bit in UB CSR 2 (on MCT module) not set after
                    ;24563  ;                   DATI to array with UB TB PAR ERR in first cycle of two cycle
                    ;24564  ;                   read.
                    ;24565  ;       Error 5 - UB CONTROL REG 2 did not have NO SSYN ERR set after DATI to
                    ;24566  ;                   array with UB PAR ERR in second cycle of two cycle read.
                    ;24567  ;       Error 6 - UB TB PAR ERR bit in UB CSR 2 (on MCT module) not set after
                    ;24568  ;                   DATI to array with UB TB PAR ERR in second cycle of two cycle
                    ;24569  ;                   read.
                    ;24570  ;
                    ;24571  ; Debug:
                    ;24572  ;
                    ;24573  ;       Error 1 - This error most likely indicates a problem with the micro-
                    ;24574  ;       code.  The UB ERRSUM branch at UB CYC RD C6 is most suspect.
                    ;24575  ;
                    ;24576  ;       Error 2 - If this is the first error, it most likely indicates that
                    ;24577  ;       the CSR ERR SUM CLK is missing from UB CYC RD C4.  Another possibility
                    ;24578  ;       is that L UB RDS H is not being cleared.  This could be caused by a
                    ;24579  ;       failure in the CSR DATA ERROR CONTROL PAL or the input CLR UBRDS L to
                    ;24580  ;       that PAL.  CLR UBRDS L should be low when the JSR [CLEAR.CSR2] in-
                    ;24581  ;       struction is executed.
                    ;24582  ;
                    ;24583  ;       Error 3 - This error most likely indicates a problem with the micro-
                    ;24584  ;       code.  The UB ERRSUM branch at UB CYC TWO ARY CYC RD C6 is most suspect.
                    ;24585  ;
                    ;24586  ;       Error 4 - This error is not likely except in conjunction with error 3.
                    ;24587  ;
                    ;24588  ;       Error 5 - This error most likely indicates a problem with the micro-
                    ;24589  ;       code.  The UB ERRSUM branch at the cycle entered by the 2ND CYC path
                    ;24590  ;       from UB CYC RD C5 is most suspect.
                    ;24591  ;
                    ;24592  ;       Error 6 - This error is not likely except in conjunction with error 5.
                    ;24593  ;
                    ;24594  ;--
                    ;24595
                    ;24596  ;       T9  = ADDRESS 1FE FOR PAGE BOUNDARY
                    ;24597  ;       T10 = ADDRESS 1FC TO WRITE TO LAST LONGWORD ON PAGE
                    ;24598  ;       T14 = DATA FOR UBE CONTROL REG 1
                    ;24599
                    ;24600  T.3F:
U 1FE1, B65E,15     ;24601          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;24602                                          ; STATUS WORD
U 1FE2, 3E80,15     ;24603          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;24604                                          ; BIT 15 INDICATES START OF TEST TO
                    ;24605                                          ; CONSOLE
U 1FE3, 10E0,15     ;24606          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;24607  WAIT.T3F.0:
U 1FE4, 09FE,44     ;24608          JMP [WAIT.T3F.0]                ;LOOP HERE WAITING FOP CONSOLE TO
                    ;24609                                          ; RESPOND
U 1FE5, 0A00,04     ;24610          JMP [T3F.START]                 ;JUMP OVER BRANCH BOUNDARY TO PREVENT
                    ;24611                                          ; VALIDITY FAILURES
                    ;24612  2000:
                    ;24613  T3F.START:
U 2000, 0A1B,8C     ;24614          JSR [ISSUE.DCLO]                ;INIT UBE
U 2001, 65A0,15     ;24615          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
```

```
U 2002, 3644,15 ;24616          MOV LS[MCT] TO WR[0]              ;SET BIT 2 IN WR
U 2003, 3E8C,15 ;24617          MOV WR[0] TO LS[MODULE.NUM]       ;SET UP TO CALL OUT MCT MODULE IF ERROR
                ;24618
U 2004, B652,15 ;24619          MOV LS[#200] TO WR[0]             ;200 IN WR
U 2005, 4142,15 ;24620          SUB LS[#2] FROM WR[0]             ;1FE IN WR
U 2006, BE12,15 ;24621          MOV WR[0] TO LS[T9]               ;STORE 1FE IN LS
U 2007, 4142,15 ;24622          SUB LS[#2] FROM WR[0]             ;1FC IN WR
U 2008, BE14,15 ;24623          MOV WR[0] TO LS[T10]              ;STORE 1FC IN LS
                ;24624
U 2009, B67A,15 ;24625          MOV LS[BIT29] TO WR[0]            ;SET BIT 29 (TB PAR DIAG) IN WR
U 200A, 8A17,FC ;24626          JSR [WRITE.CSR1]                  ;SET TB PAR DIAG BIT IN CSR 1
U 200B, 1B9D,F5 ;24627          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;24628                                            ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 200C, B27E,15 ;24629          WRITE.MEM LS[BIT31]               ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;24630                                            ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                ;24631                                            ; OFFSET CLEAR, AND PARITY ERROR
                ;24632
U 200D, 0A17,EC ;24633          JSR [CLEAR.CSR1]                  ;CLEAR CSR 1
U 200E, 367E,15 ;24634          MOV LS[BIT31] TO WR[0]            ;BIT 31 (VALID) SET IN WR
U 200F, C740,15 ;24635          BIS LS[#1] TO WR[0]               ;SET BIT TO MAP 200 TO 200
U 2010, 3E10,15 ;24636          MOV WR[0] TO LS[T8]               ;STORE IN LS
U 2011, 9B53,F5 ;24637          MEM.REQ[WRITE.UBS.MAP] ADRS[BIT9] DT[LONG]
                ;24638                                            ;SET UP TO WRITE ADDR 1 OF UNIBUS MAP
U 2012, 3210,15 ;24639          WRITE.MEM LS[T8]                  ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;24640                                            ; ADDR 200 TO PHYSICAL ADDR 200, BYTE
                ;24641                                            ; OFFSET CLR, NO PARITY ERROR
                ;24642
U 2013, 9914,75 ;24643          MEM.REQ[WRITE.P] ADRS[T10] DT[LONG]
                ;24644                                            ;SETUP TO WRITE AT ADDRESS 1FC
U 2014, B29C,15 ;24645          WRITE.MEM LS[ZERO]                ;WRITE 0'S AT ADDRESS 1FC
U 2015, 1952,75 ;24646          MEM.REQ[WRITE.P] ADRS[#200] DT[LONG]
                ;24647                                            ;SETUP TO WRITE AT ADDRESS 200
U 2016, B29C,15 ;24648          WRITE.MEM LS[ZERO]                ;WRITE 0'S AT ADDRESS 200
                ;24649
U 2017, 2F80,15 ;24650          CLR WR[0]                         ;CLEAR WR 0
U 2018, BFFE,15 ;24651          MOV WR[0] TO LS[PSL.HW]           ;CLEAR IPL BITS IN PSW
U 2019, 8A1C,7C ;24652          JSR [SETUP.DATI.NPR]              ;SETUP T14 WITH UBE DATA TO ISSUE NPR
                ;24653                                            ; DATI, INTERRUPT ON DONE AT BR6
                ;24654
U 201A, 3641,15 ;24655          MOV LS[#1] TO WR[2]               ;STARTING ERROR NUMBER (ERROR 1)
U 201B, 8A1E,3C ;24656          JSR [CLEAR.CSR2]                  ;CLEAR ERRORS IN UB CSR 2
U 201C, 8A03,AC ;24657          JSR [LOOP.T3F.2]                  ;PERFORM TEST WITH 1 CYCLE READ
                ;24658
U 201D, 367E,15 ;24659          MOV LS[BIT31] TO WR[0]            ;SET BIT 31 IN WR (VALID BIT)
U 201E, 4772,15 ;24660          BIS LS[BIT25] TO WR[0]            ;SET BIT 25 (BYTE OFFSET)
U 201F, 3E10,15 ;24661          MOV WR[0] TO LS[T8]               ;STORE IN LS
U 2020, B67A,15 ;24662          MOV LS[BIT29] TO WR[0]            ;SET BIT 29 (TB PAR DIAG) IN WR
U 2021, 8A17,FC ;24663          JSR [WRITE.CSR1]                  ;SET TB PAR DIAG BIT IN CSR 1
U 2022, 1B9D,F5 ;24664          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;24665                                            ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 2023, 3210,15 ;24666          WRITE.MEM LS[T8]                  ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;24667                                            ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                ;24668                                            ; OFFSET SET, PARITY ERROR PRESENT
U 2024, 0A17,EC ;24669          JSR [CLEAR.CSR1]                  ;CLEAR TB PAR DIAG BIT IN CSR1
                ;24670
```

```
U 2025, 4043,15 :24671          ADD LS[#2] TO WR[2]                ;STARTING ERROR NUMBER (ERROR 3)
U 2026, 8A1E,3C :24672          JSR [CLEAR.CSR2]                   ;CLEAR ERRORS IN UB CSR 2
U 2027, 8A03,AC :24673          JSR [LOOP.T3F.2]                   ;PERFORM TEST WITH 2 CYCLE READ AND
                :24674                                             ; TB PAR ERR IN FIRST CYCLE
                :24675
U 2028, 367E,15 :24676          MOV LS[BIT31] TO WR[0]             ;SET BIT 31 IN WR (VALID BIT)
U 2029, 4772,15 :24677          BIS LS[BIT25] TO WR[0]             ;SET BIT 25 (BYTE OFFSET)
U 202A, 3E10,15 :24678          MOV WR[0] TO LS[T8]                ;STORE IN LS
U 202B, 1B9D,F5 :24679          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                :24680                                             ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 202C, 3210,15 :24681          WRITE.MEM LS[T8]                   ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                :24682                                             ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                :24683                                             ; OFFSET SET, NO PARITY ERROR
                :24684
U 202D, B67A,15 :24685          MOV LS[BIT29] TO WR[0]             ;SET BIT 29 (TB PAR DIAG) IN WR
U 202E, 8A17,FC :24686          JSR [WRITE.CSR1]                   ;SET TB PAR DIAG BIT IN CSR 1
U 202F, 367E,15 :24687          MOV LS[BIT31] TO WR[0]             ;SET BIT 31 IN WR (VALID BIT)
U 2030, 4772,15 :24688          BIS LS[BIT25] TO WR[0]             ;SET BIT 25 (BYTE OFFSET)
U 2031, C740,15 :24689          BIS LS[#1] TO WR[0]                ;MAP 200 TO 200
U 2032, 3E10,15 :24690          MOV WR[0] TO LS[T8]                ;STORE IN LS
U 2033, 9B53,F5 :24691          MEM.REQ[WRITE.UBS.MAP] ADRS[BIT9] DT[LONG]
                :24692                                             ;SET UP TO WRITE ADDR 1 OF UNIBUS MAP
U 2034, 3210,15 :24693          WRITE.MEM LS[T8]                   ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                :24694                                             ; ADDR 200 TO PHYSICAL ADDR 200, BYTE
                :24695                                             ; OFFSET SET, PARITY ERROR PRESENT
U 2035, 0A17,EC :24696          JSR [CLEAR.CSR1]                   ;CLEAR CSR1
                :24697
U 2036, 4043,15 :24698          ADD LS[#2] TO WR[2]                ;STARTING ERROR NUMBER (ERROR 5)
U 2037, 8A1E,3C :24699          JSR [CLEAR.CSR2]                   ;CLEAR ERRORS IN UB CSR 2
U 2038, 8A03,AC :24700          JSR [LOOP.T3F.2]                   ;PERFORM TEST WITH 2 CYCLE READ AND
                :24701                                             ; TB PAR ERR IN SECOND CYCLE
                :24702
U 2039, 0A05,A4 :24703          JMP [END.T3F]                      ;DONE
                :24704
                :24705  ;
                :24706  ;SUBROUTINE TO PERFORM NPR TEST
                :24707  ;
                :24708  LOOP.T3F.2:
U 203A, 3E83,15 :24709          MOV WR[2] TO LS[ERROR.NUMBER]      ;SET UP ERR NUMBER IN LS (ERROR 1, 3,
                :24710                                             ; or 5)
U 203B, 5F28,15 :24711          MCOM LS[#FFFF] TO WR[0]            ;FFFF0000 IN WR
U 203C, 3E8A,15 :24712          MOV WR[0] TO LS[ERROR.MASK]        ;CHECK BITS 15-0 ONLY
                :24713  LOOP.T3F.1:
U 203D, 19B8,35 :24714          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                :24715                                             ;SET UP TO WRITE TO UBE ADDRESS REG
U 203E, B212,15 :24716          WRITE.MEM LS[T9]                   ;PUT ADDRESS 1FE IN UBE ADDRESS REG
U 203F, 99B6,35 :24717          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                :24718                                             ;SET UP TO WRITE TO UBE CYCLE COUNT
U 2040, 329E,15 :24719          WRITE.MEM LS[ONES]                 ;PUT A -1 IN CYCLE COUNT (DO 1 CYCLE)
U 2041, 99BA,35 :24720          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                :24721                                             ;SET UP TO WRITE TO UBE CONTROL REG 1
U 2042, 321C,15 :24722          WRITE.MEM LS[T14]                  ;SET UP CONTROL REG TO ISSUE NPR TO
                :24723                                             ; READ A WORD FROM ARRAY AND INTR AT
                :24724                                             ; BR6 ON DONE
                :24725  LOOP.T3F.INT.1:
```

```
U 2043, 8A04,37 ;24726          JMP.IF[NO.INTERRUPT] TO [LOOP.T3F.INT.1] ;LOOP UNTIL BR INTERRUPT
                ;24727
U 2044, 1F46,75 ;24728          MEM.REQ[ISSUE.BG] ADRS[BG6] DT[LONG]
                ;24729                                  ;SET UP TO ISSUE BUS GRANT ON LEVEL 6
U 2045, 3022,15 ;24730          MOV MEM.DATA TO WR[0]   ;COMPLETE CYCLE
U 2046, DB00,15 ;24731          NOP                     ;DELAY FOR NEXT INTR AT BR7
U 2047, 1FCE,75 ;24732          MEM.REQ[ISSUE.BG] ADRS[BG7] DT[LONG]
                ;24733                                  ;SET UP TO ISSUE BUS GRANT ON LEVEL 7
U 2048, 3022,15 ;24734          MOV MEM.DATA TO WR[0]   ;COMPLETE CYCLE
                ;24735
U 2049, B650,95 ;24736          MOV LS[NO.SSYN.ERR] TO WR[1]    ;EXPECTED DATA (NO SSYN ERR IN UBE)
U 204A, C75A,95 ;24737          BIS LS[CCOVF] TO WR[1]          ;ALSO CCOVF WILL BE SET
U 204B, 98BF,B5 ;24738          MEM.REQ[READ.P] ADRS[BECR2] DT[WORD]
                ;24739                                  ;SET UP TO READ UBE ERROR REG
U 204C, 3022,15 ;24740          MOV MEM.DATA TO WR[0]   ;GET CONTENTS
U 204D, 0869,3C ;24741          JSR [CHECK.RESULT]      ;CHECK FOR 'NO SSYN ERR' BIT SET
U 204E, 8A03,D4 ;24742          JMP [LOOP.T3F.1]        ;ERROR LOOP IF ENABLED
                ;24743
U 204F, FF82,15 ;24744          INC LS[ERROR.NUMBER]    ;ERROR 2, 4, OR 6
U 2050, 3630,15 ;24745          MOV LS[CSR2.MASK] TO WR[0]      ;MASK TO CHECK ERROR BITS IN CSR 2
U 2051, 3E8A,15 ;24746          MOV WR[0] TO LS[ERROR.MASK]     ;SET UP ERROR MASK TO CHECK ERROR BITS
U 2052, 365E,95 ;24747          MOV LS[BIT15] TO WR[1]          ;EXPECTED DATA (UB TB PARITY ERROR SET)
U 2053, 1D47,75 ;24748          MEM.REQ[READ.CSR] ADRS[CSR2] DT[LONG]
                ;24749                                  ;SET UP TO READ UB CSR 2
U 2054, 3022,15 ;24750          MOV MEM.DATA TO WR[0]   ;GET RESULT
U 2055, 0869,3C ;24751          JSR [CHECK.RESULT]      ;CHECK FOR UB TB PARITY ERROR SET
U 2056, 0A03,A4 ;24752          JMP [LOOP.T3F.2]        ;ERROR LOOP IF ENABLED
                ;24753
U 2057, 99BC,35 ;24754          MEM.REQ[WRITE.P] ADRS[CLEAR.ERR.ADDR] DT[WORD]
                ;24755                                  ;SET UP TO CLEAR UBE ERROR REG
U 2058, B29C,15 ;24756          WRITE.MEM LS[ZERO]      ;CLEAR UBE ERROR REG
U 2059, 5B00,14 ;24757          RETURN                  ;DONE WITH NPR TEST FOR THIS DATA
                ;24758
                ;24759  END.T3F:
```

```
                    ;24760  .page   "  TEST 40 ISSUE BG Sack Timeout Test (M8391 MCT Module)"
                    ;24761  ;++
                    ;24762  ;
                    ;24763  ; Test Description:
                    ;24764  ;
                    ;24765  ;       This test checks the micro-code associated with a SACK timeout when
                    ;24766  ;       a BG is issued.  A ISSUE BG will be executed without the UBE being
                    ;24767  ;       enabled.  The micro-code should take the LSACK not and TIMEOUT paths
                    ;24768  ;       and finally set the NXM bit in CSR 1.  This test checks that NXM is
                    ;24769  ;       set.
                    ;24770  ;
                    ;24771  ; Assumptions:
                    ;24772  ;
                    ;24773  ;       It is assumed that all other micro-diagnostics have run successfully.
                    ;24774  ;
                    ;24775  ; Test Steps:
                    ;24776  ;
                    ;24777  ;       1)  Set up error mask, error number, and module number in LS (for
                    ;24778  ;           error printouts) and clear previous error number in LS.  Also
                    ;24779  ;           issue DCLO to UBE to clear error bits.
                    ;24780  ;       2)  Issue BG on level 7 (level doesn't really matter).
                    ;24781  ;       3)  Read CSR 1 and check that NXM is set.
                    ;24782  ;
                    ;24783  ; Errors:
                    ;24784  ;
                    ;24785  ;       Error 1 - NXM failed to set on SACK timeout when BG issued.
                    ;24786  ;
                    ;24787  ; Debug:
                    ;24788  ;
                    ;24789  ;       Error 1 - This error most likely indicates a problem with the ISSUE BG
                    ;24790  ;       micro code.  The flow should take the LSACK not and TIMEOUT paths at
                    ;24791  ;       ISSUE BG C5 and BG TIMEOUT C1.  It should go to BG NXM, set ROT C0, and
                    ;24792  ;       issue a ERR SUM CLK to set the NXM bit in CSR 1.
                    ;24793  ;
                    ;24794  ;--
                    ;24795
                    ;24796
                    ;24797  T.40:
U 205A, B65E,15     ;24798          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;24799                                          ; STATUS WORD
U 205B, 3E80,15     ;24800          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;24801                                          ; BIT 15 INDICATES START OF TEST TO
                    ;24802                                          ; CONSOLE
U 205C, 10E0,15     ;24803          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;24804  WAIT.T40.0:
U 205D, 8A05,D4     ;24805          JMP [WAIT.T40.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;24806                                          ; RESPOND
                    ;24807
U 205E, 0A1B,8C     ;24808          JSR [ISSUE.DCLO]                ;INIT UBE
U 205F, 65A0,15     ;24809          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 2060, 3644,15     ;24810          MOV LS[MCT] TO WR[0]            ;SET BIT 2 IN WR
U 2061, 3E8C,15     ;24811          MOV WR[0] TO LS[MODULE.NUM]     ;SET UP TO CALL OUT MCT MODULE IF ERROR
U 2062, B640,15     ;24812          MOV LS[#1] TO WR[0]             ;1 IN WR
U 2063, BE82,15     ;24813          MOV WR[0] TO LS[ERROR.NUMBER]   ;ERROR 1
U 2064, 5F60,15     ;24814          MCOM LS[NXM] TO WR[0]           ;CLEAR BIT 16 IN WR
```

```
U 2065, 3E8A,15 ;24815          MOV WR[0] TO LS[ERROR.MASK]     ;SET UP ERROR MASK TO CHECK NXM BIT ONLY
                ;24816  LOOP.T40.1:
U 2066, 9FCE,35 ;24817          MEM.REQ[ISSUE.BG] ADRS[BG7] DT[WORD]
                ;24818                                          ;ISSUE BG ON LEVEL 7
U 2067, 3022,15 ;24819          MOV MEM.DATA TO WR[0]           ;COMPLETE BG CYCLE
U 2068, 9D45,75 ;24820          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;24821                                          ;SET UP TO READ CRS 1
U 2069, 3022,15 ;24822          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CSR 1
U 206A, B660,95 ;24823          MOV LS[NXM] TO WR[1]            ;EXPECTED DATA (NXM SET)
U 206B, 0869,3C ;24824          JSR [CHECK.RESULT]              ;SEE IF NXM SET
U 206C, 0A06,64 ;24825          JMP [LOOP.T40.1]                ;ERROR LOOP IF ENABLED
                ;24826
                ;24827  END.T40:
```

```
;24828  .page   "  TEST 41 Block Transfer Test (M8391 MCT Module)"
;24829  ;++
;24830  ;
;24831  ; Test Description:
;24832  ;
;24833  ;       This test checks the micro code associated with waiting for another
;24834  ;       MSYN after completing one memory cycle.  The UBE will transfer one
;24835  ;       word (DATO to array) but keep BBSY asserted.  Without issueing an-
;24836  ;       other NPR, MSYN will again be asserted in preparation for another
;24837  ;       DATO to memory.  The micro code must take the UB ACTIVE path at the
;24838  ;       end of the UNIBUS WRITE TO ARRAY routine and go to wait for another
;24839  ;       MSYN.  This will complete the test of all micro code in the UNIBUS
;24840  ;       WRITE and UNIBUS READ TO ARRAY flow.
;24841  ;
;24842  ; Assumptions:
;24843  ;
;24844  ;       It is assumed that all other micro-diagnostics have run successfully.
;24845  ;       It is also assumed that the UBE present is known to be good.
;24846  ;
;24847  ; Test Steps:
;24848  ;
;24849  ;       1)  Set up error mask, error number, and module number in LS (for
;24850  ;           error printouts) and clear previous error number in LS.  Also
;24851  ;           issue DCLO to UBE to clear error bits.
;24852  ;       2)  Set up UNIBUS MAP to map virtual address 0 to physical address 0
;24853  ;           and set the VALID bit.
;24854  ;       3)  Clear the IPL bits in the PSW (allow all interrupts).  Write all
;24855  ;           1's at physical memory address 0.
;24856  ;       4)  Set up UBE data reg with all 0's, cycle count to 1 cycle (FFFF),
;24857  ;           and address reg to address 0.
;24858  ;       5)  Set up control reg 1 of the UBE to execute block transfer DATO
;24859  ;           cycle (transfer 2 words with one NPR) via an NPR and interrupt via
;24860  ;           a BR6 when done.  Also set the GO bit to start the UBE.
;24861  ;       6)  Loop waiting for INTR.  Check for UBE error interrupt.
;24862  ;       7)  Read memory address 0 and check for all 0's (32 bits).
;24863  ;
;24864  ; Errors:
;24865  ;
;24866  ;       Error 1 - BR7 interrupt occurred (indicates UBE error).  Received data
;24867  ;                 is contents of UBE control reg 2 (error register).
;24868  ;       Error 2 - Main memory longword 0 failed to get written correctly by
;24869  ;                 UBE on double word transfer.
;24870  ;
;24871  ; Debug:
;24872  ;
;24873  ;       Error 1 - This error indicates that the UBE detected an error.  See the
;24874  ;       UBE error register list in this listing to determine what went wrong.
;24875  ;
;24876  ;       Error 2 - This error indicates that the UBE failed to write data cor-
;24877  ;       rectly in the array.  The most likely problem is with the micro code.
;24878  ;       This error is likely to occur in conjunction with error 1 above.  If
;24879  ;       error 1 indicates NO SSYN error, then check the branch on UB ACTIVE
;24880  ;       at UB CYC WR ARY C22 at the end of the UNIBUS WRITE TO ARRAY and at
;24881  ;       WAIT FOR NEXT MSYN in the UNIBUS READ TO ARRAY routine.
;24882  ;
```

```
                ;24883  ;--
                ;24884
                ;24885  ;       T14 = DATA FOR UBE CONTROL REG 1
                ;24886
                ;24887  T.41:
U 206D, B65E,15 ;24888          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;24889                                          ; STATUS WORD
U 206E, 3E80,15 ;24890          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;24891                                          ; BIT 15 INDICATES START OF TEST TO
                ;24892                                          ; CONSOLE
U 206F, 10E0,15 ;24893          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;24894  WAIT.T41.0:
U 2070, 8A07,04 ;24895          JMP [WAIT.T41.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;24896                                          ; RESPOND
                ;24897
U 2071, 0A1B,8C ;24898          JSR [ISSUE.DCLO]                ;INIT UBE
U 2072, 65A0,15 ;24899          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 2073, 3644,15 ;24900          MOV LS[MCT] TO WR[0]            ;SET BIT 2 IN WR
U 2074, 3E8C,15 ;24901          MOV WR[0] TO LS[MODULE.NUM]     ;SET UP TO CALL OUT MCT MODULE IF ERROR
U 2075, 1B9D,F5 ;24902          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;24903                                          ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 2076, B27E,15 ;24904          WRITE.MEM LS[BIT31]             ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;24905                                          ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                ;24906                                          ; OFFSET CLEAR
U 2077, 2F80,15 ;24907          CLR WR[0]                       ;CLEAR WR 0
U 2078, BFFE,15 ;24908          MOV WR[0] TO LS[PSL.HW]         ;CLEAR IPL BITS IN PSW
U 2079, 999C,75 ;24909          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;24910                                          ;SET UP TO WRITE AT MEMORY ADDR 0
U 207A, 329E,15 ;24911          WRITE.MEM LS[ONES]              ;WRITE ALL 1'S AT ADDR 0
                ;24912
U 207B, 0A1C,0C ;24913          JSR [SETUP.DATO.NPR]            ;SETUP T14 WITH UBE DATA TO ISSUE NPR
                ;24914                                          ; DATO, INTERRUPT ON DONE AT BR6
U 207C, 4554,15 ;24915          BIC LS[FUNA] TO WR[0]           ;WR 0 STILL CONTAINS UBE CONTROL DATA
                ;24916                                          ; CLEAR FUNA BIT
U 207D, 4756,15 ;24917          BIS LS[FUNB] TO WR[0]           ;SET FUNB BIT.  NOW UBE SETUP TO DO
                ;24918                                          ; DOUBLE WORD TRANSFER
U 207E, 3E1C,15 ;24919          MOV WR[0] TO LS[T14]            ;STORE FUNC IN LS
                ;24920
U 207F, 19B4,35 ;24921          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                ;24922                                          ;SET UP TO WRITE TO UBE DATA REG
U 2080, B29C,15 ;24923          WRITE.MEM LS[ZERO]              ;PUT A 0 IN DATA REG
                ;24924
                ;24925  LOOP.T41.2:
U 2081, B640,15 ;24926          MOV LS[#1] TO WR[0]             ;1 IN WR
U 2082, BE82,15 ;24927          MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
                ;24928
                ;24929  LOOP.T41.1:
U 2083, 19B8,35 ;24930          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                ;24931                                          ;SET UP TO WRITE TO UBE ADDRESS REG
U 2084, B29C,15 ;24932          WRITE.MEM LS[#0]                ;PUT A 0 IN ADDRESS REG
U 2085, 99B6,35 ;24933          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                ;24934                                          ;SET UP TO WRITE TO UBE CYCLE COUNT
U 2086, 329E,15 ;24935          WRITE.MEM LS[ONES]              ;PUT A -1 IN CYCLE COUNT (DO 1 CYCLE)
U 2087, 99BA,35 ;24936          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                ;24937                                          ;SET UP TO WRITE TO UBE CONTROL REG 1
```

```
U 2088, 321C,15 ;24938          WRITE.MEM LS[T14]                ;SET UP CONTROL REG TO ISSUE NPR TO
                ;24939                                           ; WRITE ARRAY AND INTR AT BR6 ON DONE
                ;24940  LOOP.T41.INT:
U 2089, 8A08,97 ;24941          JMP.IF[NO.INTERRUPT] TO [LOOP.T41.INT] ;LOOP UNTIL BR6 INTERRUPT
                ;24942
U 208A, 8A1C,DC ;24943          JSR [CHECK.UBE.ERR]              ;SEE IF UBE ERR AND ISSUE NECESSARY
                ;24944                                           ; GRANTS
U 208B, 8A08,34 ;24945          JMP [LOOP.T41.1]                 ;ERROR LOOP RETURN IF ENABLED
                ;24946
U 208C, FF82,15 ;24947          INC LS[ERROR.NUMBER]             ;ERROR 2
U 208D, E58A,15 ;24948          CLR LS[ERROR.MASK]               ;CHECK ALL BITS
U 208E, 189D,F5 ;24949          MEM.REQ[READ.P] ADRS[#0] DT[LONG]
                ;24950                                           ;SET UP TO READ DATA AT ADDRESS 0
U 208F, 3022,15 ;24951          MOV MEM.DATA TO WR[0]            ;GET RESULT
U 2090, 2F82,95 ;24952          CLR WR[1]                        ;EXPECTED DATA
U 2091, 0869,3C ;24953          JSR [CHECK.RESULT]               ;SEE IF ARRAY WRITTEN CORRECTLY
U 2092, 0A08,14 ;24954          JMP [LOOP.T41.2]                 ;ERROR LOOP IF ENABLED
                ;24955
                ;24956  END.T41:
```

```
;24957  .page   "  TEST 42 LOCK and UB BSY Logic Test  (M8391 MCT Module)"
;24958  ;++
;24959  ;
;24960  ; Test Description:
;24961  ;
;24962  ;       This test checks the logic associated with SET LOCK and the CSR 1 error
;24963  ;       logic used to set UB BSY.  The test sets up the UBE to do infinite NPR
;24964  ;       transfers by setting the INH INC ADDR & CC bit (bit 2) in CONTROL REG
;24965  ;       2 of the UBE.  While the UBE is doing NPR transfers, the CPU micro
;24966  ;       code is trying to do a read to UNIBUS (tries to read the UBE's cycle
;24967  ;       count reg).  If the logic is working correctly, the CPU will get an
;24968  ;       error sum sooner or later due to UNIBUS ACTIVITY being asserted.  The
;24969  ;       error will be UB BSY in CSR 1 on the MCT.  The MCT micro-code should
;24970  ;       have set LOCK when this access failed.  LOCK being set should allow
;24971  ;       the next access to gain the UNIBUS on its next try.
;24972  ;         The test uses a delay counter to continually move the second access
;24973  ;       out in time in relation to the first access.  This guarantees that the
;24974  ;       second access will fail at sometime if the set LOCK logic fails.
;24975  ;         This test will report the first error it gets and abort further test-
;24976  ;       ing.  This is necessary because the UBE must be stopped before an error
;24977  ;       is reported so that it will not keep running after a halt on error is
;24978  ;       executed.  This assures that the machine will be in a normal state.
;24979  ;       Error looping is permitted with the loop starting from the point that
;24980  ;       the UBE is setup for continuous DATO's.
;24981  ;
;24982  ; Assumptions:
;24983  ;
;24984  ;       It is assumed that all other micro-diagnostics have run successfully.
;24985  ;       It is also assumed that the UBE present is known to be good.
;24986  ;
;24987  ; Test Steps:
;24988  ;
;24989  ;       1)  Set up error mask, error number, and module number in LS (for
;24990  ;           error printouts) and clear previous error number in LS.  Also
;24991  ;           issue DCLO to UBE to clear error bits.
;24992  ;       2)  Set up UNIBUS MAP to map virtual address 0 to physical address 0
;24993  ;           and set the VALID bit.
;24994  ;       3)  Clear the IPL bits in the PSW (allow all interrupts).
;24995  ;       4)  Set up UBE data reg with alternating 1's and 0's and the address
;24996  ;           reg to address 0.
;24997  ;       5)  Set up control reg 2 of the UBE to inhibit incrementing the address
;24998  ;           and cycle count regs, so that the UBE will issue NPR transfers con-
;24999  ;           tinually once it is started.
;25000  ;       6)  Set up control reg 1 of the UBE to do DATO cycles via an NPR.  Also
;25001  ;           set the GO bit to start the UBE.
;25002  ;       7)  Decrement a timeout counter.  If this loop has not executed 1000(H)
;25003  ;           times, continue.  Else report error and abort test.
;25004  ;       8)  Execute a read to UNIBUS (READ.P to ube cycle count reg) and check
;25005  ;           for an error sum.  If error sum go to step 10.  Else repeat this
;25006  ;           step one more time.
;25007  ;       9)  Check for interrupt (UBE error).  If no interrupt, go back to step
;25008  ;           7.  If an interrupt occurs, report error and abort test.
;25009  ;       10) Read CSR 1 and check for UB BSY (bit 17) set.  If OK, continue.
;25010  ;           Else report error and abort test.
;25011  ;       11) Set up a delay counter.  This counter will be incremented each time
```

```
;25012  ;           the following loop is executed.  As the counter increments, it moves
;25013  ;           the second read to UNIBUS out 540 nanoseconds more each time in re-
;25014  ;           lation to the first read to UNIBUS failure.
;25015  ;       12) Execute 2 read to UNIBUS cycles.  Check for error sum after each.
;25016  ;           Repeat until an error sum (UB BSY) occurs, then go to step 13.
;25017  ;       13) Execute another read to UNIBUS after a delay controlled by the delay
;25018  ;           counter.  Check for no error sum (the second read is guaranteed to
;25019  ;           succeed).  If error, report and abort test.
;25020  ;       14) Repeat steps 11 through 13 until done 100(H) times.
;25021  ;
;25022  ; Errors:
;25023  ;
;25024  ;       Error 1 - Error sum never occurred when doing a read to UNIBUS when UB
;25025  ;                 ACTIVITY was asserted.
;25026  ;       Error 2 - BR7 interrupt occurred (indicates UBE error).  Received data
;25027  ;                 is contents of UBE control reg 2 (error register).
;25028  ;       Error 3 - Read to UNIBUS during NPR transfers got an error sum other
;25029  ;                 than UB BSY in CSR 1.  Expected and received data is contents
;25030  ;                 of CSR 1.
;25031  ;       Error 4 - BR7 interrupt occurred (indicates UBE error).  Received data
;25032  ;                 is contents of UBE control reg 2 (error register).
;25033  ;       Error 5 - Second attempt to read to UNIBUS resulted in error sum.  Ex-
;25034  ;                 pected and received data is contents of CSR 1.
;25035  ;
;25036  ; Debug:
;25037  ;
;25038  ;       Error 1 - This error indicates that the UB BSY logic is failing, or
;25039  ;       that LOCK is always set.  Check LOCK L at the output of the ARBITRATOR
;25040  ;       PAL while looping on error.  It should be pulsing high and low.  If not,
;25041  ;       check that ARB C0 H and ARB C1 H are both pulsing at the input to the
;25042  ;       PAL.  If not, the ARBITRATOR PAL is probably bad.
;25043  ;         If LOCK L is pulsing, check UB ACTIVITY going into the CSR 1B PAL is
;25044  ;       pulsing.  If not, trace it back to the ARBITRATOR PAL where is is gen-
;25045  ;       erated.  If UB ACTIVITY is OK, check that P ERR SUM L and UBBSY H are
;25046  ;       pulsing at the output.  The CSR 1B PAL is probably bad if not.
;25047  ;
;25048  ;       Error 2 - This error indicates that the UBE detected an error.  See the
;25049  ;       UBE error register list in this listing to determine what went wrong.
;25050  ;
;25051  ;       Error 3 - This error indicates an error sum other than UB BSY occured
;25052  ;       on a read to the UNIBUS while NPR transfers were taking place from the
;25053  ;       UBE.  The READ should have been aborted via a UB BSY.  Determine from
;25054  ;       the MCT CSR 1 error bits which error occurred.  If no bits are set, the
;25055  ;       CSR 1B PAL is probably not setting the output UB BSY, and the PAL should
;25056  ;       be replaced (see error 1 above).
;25057  ;
;25058  ;       Error 4 - This error indicates that the UBE detected an error.  See the
;25059  ;       UBE error register list in this listing to determine what went wrong.
;25060  ;
;25061  ;       Error 5 - This is the most likely error to occur in this test.  It most
;25062  ;       likely indicates that SET LOCK failed to lock out NPG on the second
;25063  ;       read to UNIBUS request.  Check the signal LOCK L out of the ARBITRATOR
;25064  ;       PAL while looping on error.  It should be pulsing low.  If not, check
;25065  ;       the input ARB C0 H and ARB C1 H.  ARB C0 H should be going high while
;25066  ;       ARB C1 high is low during the error loop.  If it is, the ARBITRATOR PAL
```

```
                ;25067  ;       is probably bad.
                ;25068  ;
                ;25069  ;--
                ;25070
                ;25071  ;       T10 = LOOP COUNTER TO REPEAT TEST 100(H) TIMES
                ;25072  ;       T11 = TIMEOUT LOOP COUNTER IN CASE UB BSY DOESN'T WORK
                ;25073  ;       T12 = DELAY LOOP COUNTER USED TO MOVE THE SECOND READ TO UNIBUS OUT
                ;25074  ;             540 NANOSECONDS MORE EACH TIME THROUTH THE LOOP.
                ;25075  ;       T14 = DATA FOR UBE CONTROL REG 1
                ;25076
                ;25077  T.42:
U 2093, B65E,15 ;25078          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                ;25079                                          ; STATUS WORD
U 2094, 3E80,15 ;25080          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                ;25081                                          ; BIT 15 INDICATES START OF TEST TO
                ;25082                                          ; CONSOLE
U 2095, 10E0,15 ;25083          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;25084  WAIT.T42.0:
U 2096, 0A09,64 ;25085          JMP [WAIT.T42.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                ;25086                                          ; RESPOND
                ;25087
U 2097, 0A1B,8C ;25088          JSR [ISSUE.DCLO]                ;INIT UBE
U 2098, 65A0,15 ;25089          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 2099, 3644,15 ;25090          MOV LS[MCT] TO WR[0]            ;SET BIT 2 IN WR
U 209A, 3E8C,15 ;25091          MOV WR[0] TO LS[MODULE.NUM]     ;SET UP TO CALL OUT MCT MODULE IF ERROR
U 209B, 1B9D,F5 ;25092          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                ;25093                                          ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 209C, B27E,15 ;25094          WRITE.MEM LS[BIT31]             ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                ;25095                                          ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                ;25096                                          ; OFFSET CLEAR
                ;25097
U 209D, 2F80,15 ;25098          CLR WR[0]                       ;CLEAR WR 0
U 209E, BFFE,15 ;25099          MOV WR[0] TO LS[PSL.HW]         ;CLEAR IPL BITS IN PSW
                ;25100
U 209F, 0A1C,0C ;25101          JSR [SETUP.DATO.NPR]            ;SETUP T14 WITH UBE DATA TO ISSUE NPR
                ;25102                                          ; DATO, INTERRUPT ON DONE AT BR6
U 20A0, 454C,15 ;25103          BIC LS[INT.ODNE] TO WR[0]       ;CLEAR INTERRUPT ON DONE BIT
U 20A1, 4546,15 ;25104          BIC LS[BR6] TO WR[0]            ;CLEAR BR6 INTERRUPT REQUEST
U 20A2, 3E1C,15 ;25105          MOV WR[0] TO LS[T14]            ;STORE DATA FOR UBE CR1 IN LS
                ;25106
U 20A3, 999C,75 ;25107          MEM.REQ[WRITE.P] ADRS[#0] DT[LONG]
                ;25108                                          ;SET UP TO WRITE 0'S AT ADDRESS 0
U 20A4, B29C,15 ;25109          WRITE.MEM LS[ZERO]              ;WRITE 0'S IN MEMORY
                ;25110
U 20A5, 19B4,35 ;25111          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                ;25112                                          ;SET UP TO WRITE TO UBE DATA REG
U 20A6, B29A,15 ;25113          WRITE.MEM LS[#55555555]         ;PUT ALTERNATING 1'S AND 0'S IN DATA REG
U 20A7, 369A,15 ;25114          MOV LS[#55555555] TO WR[0]      ;STORE IN WR 0
U 20A8, 0A0A,EC ;25115          JSR [LOOP.T42.1.3]              ;PERFORM TEST WITH DATA OF 5555
                ;25116
U 20A9, 19BE,35 ;25117          MEM.REQ[WRITE.P] ADRS[BECR2] DT[WORD]
                ;25118                                          ;SET UP TO WRITE CONTROL REG 2
U 20AA, B29C,15 ;25119          WRITE.MEM LS[ZERO]              ;CLEAR UBE CR2 IF UB NOT BSY
U 20AB, 19BE,35 ;25120          MEM.REQ[WRITE.P] ADRS[BECR2] DT[WORD]
                ;25121                                          ;SET UP TO WRITE CONTROL REG 2 AGAIN
```

```
                    ;25122                                  ; IN CASE UB BSY DURING WRITE
U 20AC, B29C,15     ;25123          WRITE.MEM LS[ZERO]      ;COMPLETE CYCLE TO CLEAR UBE CR2 AND
                    ;25124                                  ; LOCK IF SET
U 20AD, 8A11,74     ;25125          JMP [END.T42]           ;DONE
                    ;25126
                    ;25127  ;
                    ;25128  ;SUBROUTINE TO PERFORM NPR TEST
                    ;25129  ;
                    ;25130
                    ;25131  LOOP.T42.1.3:
U 20AE, 3650,15     ;25132          MOV LS[#100] TO WR[0]   ;SET UP 100(H) IN WR
U 20AF, BE14,15     ;25133          MOV WR[0] TO LS[T10]    ;SET UP LOOP COUNTER IN LS
U 20B0, B658,15     ;25134          MOV LS[#1000] TO WR[0]  ;SET UP 1000(H) IN WR
U 20B1, 3E16,15     ;25135          MOV WR[0] TO LS[T11]    ;SET UP TIMEOUT COUNTER IN LS
U 20B2, 6518,15     ;25136          CLR LS[T12]             ;INIT DELAY COUNTER WHICH MOVES SECOND
                    ;25137                                  ; READ TO UNIBUS OUT 540 NANO-SECONDS
                    ;25138                                  ; MORE EACH TIME THROUGH THE LOOP
U 20B3, E580,15     ;25139          CLR LS[CONTROL.STATUS]  ;CLEAR CONTROL AND STATUS REG
                    ;25140
U 20B4, 19B8,35     ;25141          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                    ;25142                                  ;SET UP TO WRITE TO UBE ADDRESS REG
U 20B5, B29C,15     ;25143          WRITE.MEM LS[#0]        ;PUT A 0 IN ADDRESS REG
U 20B6, 99B6,35     ;25144          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                    ;25145                                  ;SET UP TO WRITE TO UBE CYCLE COUNT
U 20B7, 329E,15     ;25146          WRITE.MEM LS[ONES]      ;PUT A -1 IN CYCLE COUNT
                    ;25147
U 20B8, 19BE,35     ;25148          MEM.REQ[WRITE.P] ADRS[BECR2] DT[WORD]
                    ;25149                                  ;SET UP TO WRITE TO UBE CONTROL REG 2
U 20B9, B244,15     ;25150          WRITE.MEM LS[INH.INC.ADDR&CC]   ;SET UP CONTROL REG TO INHIBIT INC OF
                    ;25151                                  ; ADDRESS AND CYCLE COUNT REGS (WILL
                    ;25152                                  ; KEEP ISSUING NPR XFERS TIL STOPPED)
U 20BA, 99BA,35     ;25153          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                    ;25154                                  ;SET UP TO WRITE TO UBE CONTROL REG 1
U 20BB, 321C,15     ;25155          WRITE.MEM LS[T14]       ;SET UP CONTROL REG TO ISSUE NPR TO
                    ;25156                                  ; WRITE ARRAY
                    ;25157  REPEAT.T42.1.3:
U 20BC, B640,15     ;25158          MOV LS[#1] TO WR[0]     ;1 IN WR
U 20BD, BE82,15     ;25159          MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
U 20BE, E51A,15     ;25160          CLR LS[T13]             ;DELAY COUNTER FOR UBBSY ERROR
                    ;25161  REPEAT.T42.1:
U 20BF, 7F1A,15     ;25162          INC LS[T13]             ;INC DELAY COUNTER
U 20C0, B61B,95     ;25163          MOV LS[T13] TO WR[3]    ;STORE IN WR[3]
                    ;25164          DEC LS[T11],            ;DECREMENT TIMEOUT COUNTER
U 20C1, FC16,35     ;25165           DT(LONG)&SET.ALU.CC    ; AND SET CONDITION CODES
U 20C2, 8A0F,A9     ;25166          JMP.IF[EQL] TO [ERROR.T42.1]    ;REPORT NO ERROR SUM ERROR AND ABORT
                    ;25167                                  ; IF NO LOOP ON ERROR
                    ;25168  DELAY.T42.1:
                    ;25169          DEC WR[3],              ;DECREMENT DELAY COUNTER
U 20C3, A107,B5     ;25170           DT(LONG)&SET.ALU.CC    ; AND SET CONDITION CODES
U 20C4, 0A0C,31     ;25171          JMP.IF[NEQ] TO [DELAY.T42.1]    ;LOOP UNTIL DELAY GOES TO 0
                    ;25172
U 20C5, 98BF,B5     ;25173          MEM.REQ[READ.P] ADRS[BECR2] DT[WORD]
                    ;25174                                  ;TRY TO READ UBE CONTROL REG 2
U 20C6, 3022,15     ;25175          MOV MEM.DATA TO WR[0]   ;COMPLETE CYCLE
U 20C7, 5B00,1D     ;25176          SKIP.IF[MEM.REF.OK]     ;SKIP IF NO ERROR SUM YET
```

```
U 20C8, 0A0D,24 ;25177          JMP [CHECK.UB.BSY]              ;IF ERRSUM, SEE IF UB BSY SET IN CSR1
U 20C9, 98BF,B5 ;25178          MEM.REQ[READ.P] ADRS[BECR2] DT[WORD]
                ;25179                                          ;ELSE TRY TO READ UBE CONTROL REG 2 AGAIN
U 20CA, 3022,15 ;25180          MOV MEM.DATA TO WR[0]           ;COMPLETE CYCLE
U 20CB, 5B00,1D ;25181          SKIP.IF[MEM.REF.OK]             ;SKIP IF NO ERROR SUM YET
U 20CC, 0A0D,24 ;25182          JMP [CHECK.UB.BSY]              ;IF ERRSUM, SEE IF UB BSY SET IN CSR1
U 20CD, 8A0B,F7 ;25183          JMP.IF[NO.INTERRUPT] TO [REPEAT.T42.1]
                ;25184                                          ; ELSE REPEAT IF NO UBE ERROR (UBE WILL
                ;25185                                          ; INTERRUPT ON BR7 IF ERROR)
U 20CE, FF82,15 ;25186          INC LS[ERROR.NUMBER]            ;ERROR 2
U 20CF, 8A1C,DC ;25187          JSR [CHECK.UBE.ERR]             ;IF INTERRUPT, CHECK FOR UBE ERROR AND
                ;25188                                          ; ABORT TEST IF NO LOOP ON ERROR
U 20D0, 8A0A,E4 ;25189          JMP [LOOP.T42.1.3]              ;ERROR LOOP RETURN
U 20D1, 5B00,14 ;25190          RETURN                          ;NO ERROR LOOP RETURN. ABORT TEST
                ;25191  CHECK.UB.BSY:
U 20D2, 36C0,15 ;25192          MOV LS[#3(H)] TO WR[0]          ;PUT A 3 IN WR
U 20D3, BE82,15 ;25193          MOV WR[0] TO LS[ERROR.NUMBER]   ;ERROR 3
U 20D4, 362E,15 ;25194          MOV LS[CSR1.MASK] TO WR[0]      ;SET UP ERROR MASK FOR CSR1
U 20D5, 3E8A,15 ;25195          MOV WR[0] TO LS[ERROR.MASK]     ;STORE IN LS
U 20D6, 9D45,75 ;25196          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;25197                                          ;SET UP TO READ CSR 1
U 20D7, 3022,15 ;25198          MOV MEM.DATA TO WRL0]           ;GET RESULT
U 20D8, 458A,15 ;25199          BIC LS[ERROR.MASK] TO WR[0]     ;CLEAR ALL BUT ERROR BITS IN CSR 1 DATA
                ;25200          CMP LS[UB.BSY] WITH WR[0],      ;SEE IF UB.BSY SET, OTHERS CLEAR
U 20D9, 4E62,35 ;25201           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 20DA, 0A10,41 ;25202          JMP.IF[NEQ] TO [ERROR.T42.3]    ;REPORT ERROR IF INCORRECT AND ABORT
                ;25203                                          ; TEST IF NO LOOP ON ERROR
                ;25204  CHECK.T42.NO.ERR:
U 20DB, 3644,15 ;25205          MOV LS[#4] TO WR[0]             ;SET UP 4 IN WR
U 20DC, BE82,15 ;25206          MOV WR[0] TO LS[ERROR.NUMBER]   ;ERROR 4
U 20DD, FF18,15 ;25207          INC LS[T12]                     ;INC DELAY COUNTER
U 20DE, B619,15 ;25208          MOV LS[T12] TO WR[2]            ;USE WR AS COUNTER
                ;25209  REPEAT.T42.4:
U 20DF, B618,95 ;25210          MOV LS[T13] TO WR[3]            ;STORE DELAY COUNT IN WR[3]
U 20E0, DB00,15 ;25211          NOP
U 20E1, DB00,15 ;25212          NOP
U 20E2, DB00,15 ;25213          NOP                             ;THESE NOPS MAKE TWO LOOPS SAME SIZE
                ;25214
                ;25215  DELAY.T42.4:
                ;25216          DEC WR[3],                      ;DECREMENT DELAY COUNTER
U 20E3, A107,B5 ;25217           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 20E4, 8A0E,31 ;25218          JMP.IF[NEQ] TO [DELAY.T42.4]    ;LOOP UNTIL DELAY GOES TO 0
                ;25219
U 20E5, 98BF,B5 ;25220          MEM.REQ[READ.P] ADRS[BECR2] DT[WORD]
                ;25221                                          ;TRY TO READ UBE CONTROL REG 2
U 20E6, 3022,15 ;25222          MOV MEM.DATA TO WR[0]           ;COMPLETE CYCLE
U 20E7, 5B00,1D ;25223          SKIP.IF[MEM.REF.OK]             ;SKIP IF NO ERROR SUM YET
U 20E8, 8A0F,14 ;25224          JMP [CHECK.T42.SECOND.ACCESS]   ;IF ERROR SUM, TRY SECOND ACCESS
U 20E9, 98BF,B5 ;25225          MEM.REQ[READ.P] ADRS[BECR2] DT[WORD]
                ;25226                                          ;ELSE TRY TO READ UBE CONTROL REG 2 AGAIN
U 20EA, 3022,15 ;25227          MOV MEM.DATA TO WR[0]           ;COMPLETE CYCLE
U 20EB, 5B00,1D ;25228          SKIP.IF[MEM.REF.OK]             ;SKIP IF NO ERROR SUM YET
U 20EC, 8A0F,14 ;25229          JMP [CHECK.T42.SECOND.ACCESS]   ;IF ERROR SUM, TRY SECOND ACCESS
U 20ED, 8A0D,F7 ;25230          JMP.IF[NO.INTERRUPT] TO [REPEAT.T42.4]
                ;25231                                          ; ELSE REPEAT IF NO UBE ERROR (UBE WILL
```

```
                ;25232                                  ; INTERRUPT ON BR7 IF ERROR)
U 20EE, 8A1C,DC ;25233          JSR [CHECK.UBE.ERR]     ;IF INTERRUPT, CHECK FOR UBE ERROR AND
                ;25234                                  ; ABORT TEST IF NO LOOP ON ERROR
U 20EF, 8A0A,E4 ;25235          JMP [LOOP.T42.1.3]      ;ERROR LOOP RETURN
U 20F0, 5B00,14 ;25236          RETURN                  ;NO ERROR LOOP RETURN. ABORT TEST
                ;25237
                ;25238  CHECK.T42.SECOND.ACCESS:
                ;25239          DEC WR[2],              ;DECREMENT DELAY COUNTER
U 20F1, A105,35 ;25240           DT(LONG)&SET.ALU.CC    ; AND SET CONDITION CODES
U 20F2, 8A0F,11 ;25241          JMP.IF[NEQ] TO [CHECK.T42.SECOND.ACCESS]
                ;25242                                  ;REPEAT TIL COUNT EXHAUSTED
U 20F3, 98BF,B5 ;25243          MEM.REQ[READ.P] ADRS[BECR2] DT[WORD]
                ;25244                                  ;TRY TO READ UBE CONTROL REG 2
U 20F4, 3022,15 ;25245          MOV MEM.DATA TO WR[0]   ;COMPLETE CYCLE
U 20F5, 5B00,1D ;25246          SKIP.IF[MEM.REF.OK]     ;SKIP IF NO ERROR SUM THIS TIME
                ;25247
U 20F6, 8A10,C4 ;25248          JMP [ERROR.T42.5]       ;ELSE REPORT ERROR SUM FAILURE
                ;25249  DEC.T42.COUNTER:
                ;25250          DEC LS[T10],            ;DECREMENT COUNTER
U 20F7, 7C14,35 ;25251           DT(LONG)&SET.ALU.CC    ; AND SET CONDITION CODES
U 20F8, 0A0D,B1 ;25252          JMP.IF[NEQ] TO [CHECK.T42.NO.ERR] ;REPEAT UNTIL DONE 100(H) TIMES
                ;25253
U 20F9, 5B00,14 ;25254          RETURN                  ;DONE
                ;25255
                ;25256  ERROR.T42.1:
U 20FA, B66E,15 ;25257          MOV LS[NA.EXP.REC] TO WR[0]     ;SET BIT IN WR
U 20FB, 3E80,15 ;25258          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL AND STATUS WORD TO
                ;25259                                  ; PRINT N/A UNDER EXP AND REC DATA
U 20FC, 19BE,35 ;25260          MEM.REQ[WRITE.P] ADRS[BECR2] DT[WORD]
                ;25261                                  ;SET UP TO WRITE CONTROL REG 2
U 20FD, B29C,15 ;25262          WRITE.MEM LS[ZERO]      ;CLEAR UBE CR2 IF UB NOT BSY
U 20FE, 19BE,35 ;25263          MEM.REQ[WRITE.P] ADRS[BECR2] DT[WORD]
                ;25264                                  ;SET UP TO WRITE CONTROL REG 2 AGAIN
                ;25265                                  ; IN CASE UB BSY DURING WRITE
U 20FF, B29C,15 ;25266          WRITE.MEM LS[ZERO]      ;COMPLETE CYCLE TO CLEAR UBE CR2 AND
                ;25267                                  ; LOCK IF SET
U 2100, 369E,95 ;25268          MOV LS[ONES] TO WR[1]   ;FAKE ERROR
U 2101, 0869,3C ;25269          JSR [CHECK.RESULT]      ;REPORT ERROR 1
U 2102, 8A0A,E4 ;25270          JMP [LOOP.T42.1.3]      ;ERROR LOOP IF ENABLED
U 2103, 5B00,14 ;25271          RETURN                  ;ABORT TEST
                ;25272
                ;25273
                ;25274  ERROR.T42.3:
U 2104, 3662,95 ;25275          MOV LS[UB.BSY] TO WR[1] ;EXPECTED DATA
                ;25276                                  ; CSR1 DATA IS ALREADY IN WR[0]
U 2105, 19BE,35 ;25277          MEM.REQ[WRITE.P] ADRS[BECR2] DT[WORD]
                ;25278                                  ;SET UP TO WRITE CONTROL REG 2
U 2106, B29C,15 ;25279          WRITE.MEM LS[ZERO]      ;CLEAR UBE CR2 IF UB NOT BSY
U 2107, 19BE,35 ;25280          MEM.REQ[WRITE.P] ADRS[BECR2] DT[WORD]
                ;25281                                  ;SET UP TO WRITE CONTROL REG 2 AGAIN
                ;25282                                  ; IN CASE UB BSY DURING WRITE
U 2108, B29C,15 ;25283          WRITE.MEM LS[ZERO]      ;COMPLETE CYCLE TO CLEAR UBE CR2 AND
                ;25284                                  ; LOCK IF SET
U 2109, 0869,3C ;25285          JSR [CHECK.RESULT]      ;SEE WHICH ERROR BIT IS SET IN MCT CSR
U 210A, 8A0A,E4 ;25286          JMP [LOOP.T42.1.3]      ;ERROR LOOP IF ENABLED
```

```
U 210B, 5B00,14 ;25287          RETURN                          ;ABORT TEST
                ;25288
                ;25289  ERROR.T42.5:
U 210C, FF82,15 ;25290          INC LS[ERROR.NUMBER]            ;ERROR 5
U 210D, 2F82,95 ;25291          CLR WR[1]                       ;EXPECTED DATA
U 210E, 9D45,75 ;25292          MEM.REQ[READ.CSR] ADRS[CSR1] DT[LONG]
                ;25293                                          ;SET UP TO READ CSR 1
U 210F, 3022,15 ;25294          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 2110, 19BE,35 ;25295          MEM.REQ[WRITE.P] ADRS[BECR2] DT[WORD]
                ;25296                                          ;SET UP TO WRITE CONTROL REG 2
U 2111, B29C,15 ;25297          WRITE.MEM LS[ZERO]              ;CLEAR UBE CR2 IF UB NOT BSY
U 2112, 19BE,35 ;25298          MEM.REQ[WRITE.P] ADRS[BECR2] DT[WORD]
                ;25299                                          ;SET UP TO WRITE CONTROL REG 2 AGAIN
                ;25300                                          ; IN CASE UB BSY DURING WRITE
U 2113, B29C,15 ;25301          WRITE.MEM LS[ZERO]              ;COMPLETE CYCLE TO CLEAR UBE CR2 AND
                ;25302                                          ; LOCK IF SET
U 2114, 0869,3C ;25303          JSR [CHECK.RESULT]              ;SEE WHICH ERROR BIT IS SET IN MCT CSR
U 2115, 8A0A,E4 ;25304          JMP [LOOP.T42.1.3]              ;ERROR LOOP IF ENABLED
U 2116, 5B00,14 ;25305          RETURN                          ;ABORT TEST
                ;25306
                ;25307  END.T42:
```

```
;25308  .page   "  Test 43 UBE Multiple Write/Read To Array Test  (M8391 MCT Module)"
;25309  ;++
;25310  ;
;25311  ; Test Description:
;25312  ;
;25313  ;       This test checks the UNIBUS logic at speed by doing 1 page (256 words)
;25314  ;       of writes by the UBE to the array and checking the results in memory.
;25315  ;       Data used is first alternating 1's and 0's and then alternating 0's and
;25316  ;       1's.
;25317  ;
;25318  ; Assumptions:
;25319  ;
;25320  ;       It is assumed that all other micro-diagnostics have run successfully.
;25321  ;       It is also assumed that the UBE present is known to be good.
;25322  ;
;25323  ; Test Steps:
;25324  ;
;25325  ;       1)  Set up error mask, error number, and module number in LS (for
;25326  ;           error printouts) and clear previous error number in LS.  Also
;25327  ;           issue DCLO to UBE to clear error bits.
;25328  ;       2)  Set up UNIBUS MAP to map virtual address 0 to physical address 0
;25329  ;           and set the VALID bit.
;25330  ;       3)  Clear the IPL bits in the PSW (allow all interrupts).
;25331  ;       4)  Set up UBE data reg with alternating 1's and 0's, cycle count to
;25332  ;           256 cycles (complement of 100(H)), and address reg to address 0.
;25333  ;       5)  Set up control reg 1 of the UBE to DATO cycles via an NPR and
;25334  ;           interrupt via a BR6 when done with 1024 transfers.  Also set the
;25335  ;           GO bit to start the UBE.
;25336  ;       6)  Loop waiting for BR interrupt.
;25337  ;       7)  Check the identifier code when BR interrupt occurs.  If a BR 7 was
;25338  ;           received (indicates a UBE error), read the UBE error register and
;25339  ;           report the error.
;25340  ;       8)  Read memory addresses 0 - 1FF and check for correct data.
;25341  ;       9)  Repeat steps 4 through 8 with complemented data.
;25342  ;
;25343  ; Errors:
;25344  ;
;25345  ;       Error 1 - BR7 interrupt occurred (indicates UBE error).  Received data
;25346  ;                 is contents of UBE control reg 2 (error register).  OTHER
;25347  ;                 data is contents of UBE address reg (last address written+2).
;25348  ;       Error 2 - Main memory not written correctly by UBE.  Address under
;25349  ;                 OTHER.
;25350  ;
;25351  ; Debug:
;25352  ;
;25353  ;       Error 1 - This error indicates that the UBE detected an error.  See the
;25354  ;       UBE error register list in this listing to determine what went wrong.
;25355  ;
;25356  ;       Error 2 - This error indicates that the UBE failed to write data cor-
;25357  ;       rectly in the array.  All the hardware used has been tested in previous
;25358  ;       tests, but this is the first time continous UNIBUS activity is tested.
;25359  ;       An error probably indicates a problem with timing.
;25360  ;--
;25361
;25362  ;       T13 = CYCLE COUNT (2'S COMP OF 256 DECIMAL)
```

```
                    ;25363  ;       T14 = DATA FOR UBE CONTROL REG 1
                    ;25364  ;       T15 = EXPECTED ARRAY DATA
                    ;25365
                    ;25366  T.43:
U 2117, B65E,15     ;25367          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;25368                                          ; STATUS WORD
U 2118, 3E80,15     ;25369          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;25370                                          ; BIT 15 INDICATES START OF TEST TO
                    ;25371                                          ; CONSOLE
U 2119, 10E0,15     ;25372          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;25373  WAIT.T43.0:
U 211A, 0A11,A4     ;25374          JMP [WAIT.T43.0]                ;LOOP HERE WAITING FOR CONSOLE TO
                    ;25375                                          ; RESPOND
                    ;25376
U 211B, 0A1B,8C     ;25377          JSR [ISSUE.DCLO]                ;INIT UBE
U 211C, 65A0,15     ;25378          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 211D, 3644,15     ;25379          MOV LS[MCT] TO WR[0]            ;SET BIT 2 IN WR
U 211E, 3E8C,15     ;25380          MOV WR[0] TO LS[MODULE.NUM]     ;SET UP TO CALL OUT MCT MODULE IF ERROR
U 211F, 1B9D,F5     ;25381          MEM.REQ[WRITE.UBS.MAP] ADRS[#0] DT[LONG]
                    ;25382                                          ;SET UP TO WRITE ADDR 0 OF UNIBUS MAP
U 2120, B27E,15     ;25383          WRITE.MEM LS[BIT31]             ;SET VALID BIT IN UB MAP, MAP VIRTUAL
                    ;25384                                          ; ADDR 0 TO PHYSICAL ADDR 0, BYTE
                    ;25385                                          ; OFFSET CLEAR
U 2121, 2F80,15     ;25386          CLR WR[0]                       ;CLEAR WR 0
U 2122, BFFE,15     ;25387          MOV WR[0] TO LS[PSL.HW]          ;CLEAR IPL BITS IN PSW
                    ;25388
U 2123, 0A1C,0C     ;25389          JSR [SETUP.DATO.NPR]            ;SETUP T14 WITH UBE DATA TO ISSUE NPR
                    ;25390                                          ; DATO, INTERRUPT ON DONE AT BR6
                    ;25391
U 2124, DE50,15     ;25392          MNEG LS[#100] TO WR[0]          ;2'S COMPLEMENT OF 100(H) IN WR
U 2125, 3E1A,15     ;25393          MOV WR[0] TO LS[T13]             ;STORE IN LS
U 2126, B670,15     ;25394          MOV LS[OTHER.DATA] TO WR[0]      ;SET BIT TO PRINT UNDER OTHER DATA
U 2127, 3E80,15     ;25395          MOV WR[0] TO LS[CONTROL.STATUS] ;WRITE IN CONTROL AND STATUS WORD
                    ;25396
U 2128, 6512,15     ;25397          CLR LS[T9]                      ;CLEAR STARTING ADDRESS FOR BACKGROUND
                    ;25398                                          ; WRITE
                    ;25399  REPEAT.T43.BACK:
U 2129, 9912,75     ;25400          MEM.REQ[WRITE.P] ADRS[T9] DT[LONG]
                    ;25401                                          ;SET UP TO WRITE BACKGROUND AT CURRENT
                    ;25402                                          ; ADDRESS
U 212A, B29C,15     ;25403          WRITE.MEM LS[ZERO]              ;WRITE 0'S IN MEMORY
                    ;25404
U 212B, 3612,15     ;25405          MOV LS[T9] TO WR[0]             ;GET CURRENT ADDRESS
U 212C, C044,15     ;25406          ADD LS[#4] TO WR[0]             ;INCREMENT ADDRESS BY 4
U 212D, BE12,15     ;25407          MOV WR[0] TO LS[T9]             ;UPDATE ADDRESS IN LS
                    ;25408          CMP LS[#200] WITH WR[0],        ;SEE IF DONE 1 PAGE
U 212E, 4E52,35     ;25409           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 212F, 0A12,91     ;25410          JMP.IF[NEQ] TO [REPEAT.T43.BACK] ;REPEAT UNTIL 1 PAGE WRITTEN
                    ;25411
U 2130, 19B4,35     ;25412          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                    ;25413                                          ;SET UP TO WRITE TO UBE DATA REG
U 2131, B29A,15     ;25414          WRITE.MEM LS[#55555555]         ;PUT ALTERNATING 1'S AND 0'S IN DATA REG
U 2132, 369A,15     ;25415          MOV LS[#55555555] TO WR[0]      ;STORE IN WR 0
U 2133, BE1E,15     ;25416          MOV WR[0] TO LS[T15]             ;EXPECTED DATA
U 2134, 8A13,BC     ;25417          JSR [LOOP.T43.1.2]              ;PERFORM TEST WITH DATA OF 5555
```

```
                    ;25418
U 2135, 19B4,35     ;25419          MEM.REQ[WRITE.P] ADRS[BEDB] DT[WORD]
                    ;25420                                          ;SET UP TO WRITE TO UBE DATA REG
U 2136, 3298,15     ;25421          WRITE.MEM LS[#AAAAAAAA]         ;PUT ALTERNATING 0'S AND 1'S IN DATA REG
U 2137, B698,15     ;25422          MOV LS[#AAAAAAAA] TO WR[0]      ;STORE IN WR 0
U 2138, BE1E,15     ;25423          MOV WR[0] TO LS[T15]            ;EXPECTED DATA
U 2139, 8A13,BC     ;25424          JSR [LOOP.T43.1.2]              ;PERFORM TEST WITH DATA OF AAAA
                    ;25425
U 213A, 0A15,B4     ;25426          JMP [END.T43]                   ;DONE
                    ;25427
                    ;25428  ;
                    ;25429  ;SUBROUTINE TO PERFORM NPR TEST
                    ;25430  ;
                    ;25431
                    ;25432  LOOP.T43.1.2:
U 213B, B640,15     ;25433          MOV LS[#1] TO WR[0]             ;1 IN WR
U 213C, BE82,15     ;25434          MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
                    ;25435
U 213D, 19B8,35     ;25436          MEM.REQ[WRITE.P] ADRS[BEBA] DT[WORD]
                    ;25437                                          ;SET UP TO WRITE TO UBE ADDRESS REG
U 213E, B29C,15     ;25438          WRITE.MEM LS[#0]                ;PUT A 0 IN ADDRESS REG
U 213F, 99B6,35     ;25439          MEM.REQ[WRITE.P] ADRS[BECC] DT[WORD]
                    ;25440                                          ;SET UP TO WRITE TO UBE CYCLE COUNT
U 2140, 321A,15     ;25441          WRITE.MEM LS[T13]               ;PUT COMP OF 100(H) IN CYCLE COUNT (DO
                    ;25442                                          ; 256 CYCLES)
U 2141, 99BA,35     ;25443          MEM.REQ[WRITE.P] ADRS[BECR1] DT[WORD]
                    ;25444                                          ;SET UP TO WRITE TO UBE CONTROL REG 1
U 2142, 321C,15     ;25445          WRITE.MEM LS[T14]               ;SET UP CONTROL REG TO ISSUE NPR TO
                    ;25446                                          ; WRITE ARRAY AND INTR AT BR6 ON DONE
                    ;25447  LOOP.T43.INT:
U 2143, 0A14,37     ;25448          JMP.IF[NO.INTERRUPT] TO [LOOP.T43.INT] ;WAIT FOR BR INTERRUPT
                    ;25449
U 2144, 98B9,B5     ;25450          MEM.REQ[READ.P] ADRS[BEBA] DT[WORD]
                    ;25451                                          ;SET UP TO READ UBE ADDRESS REG IN CASE
                    ;25452                                          ; ERROR OCCURS
U 2145, 3022,15     ;25453          MOV MEM.DATA TO WR[0]           ;GET CONTENTS IF UBE ADDR REG
U 2146, BE88,15     ;25454          MOV WR[0] TO LS[ADDRESS.DATA]   ;STORE UNDER OTHER DATA
U 2147, 8A1C,DC     ;25455          JSR [CHECK.UBE.ERR]             ;SEE IF UBE ERR AND ISSUE NECESSARY
                    ;25456                                          ; GRANTS
U 2148, 0A13,B4     ;25457          JMP [LOOP.T43.1.2]              ;ERROR LOOP RETURN IF ENABLED
                    ;25458
U 2149, FF82,15     ;25459          INC LS[ERROR.NUMBER]            ;ERROR 2
U 214A, 6588,15     ;25460          CLR LS[ADDRESS.DATA]            ;0 IN OTHER DATA (ADDRESS BEING TESTED)
U 214B, E58A,15     ;25461          CLR LS[ERROR.MASK]              ;CHECK ALL BITS
U 214C, 6512,15     ;25462          CLR LS[T9]                      ;STARTING ADDRESS
                    ;25463  REPEAT.T43.READ:
U 214D, 1813,F5     ;25464          MEM.REQ[READ.P] ADRS[T9] DT[LONG]
                    ;25465                                          ;SET UP TO READ DATA AT CURRENT ADDRESS
U 214E, 3022,15     ;25466          MOV MEM.DATA TO WR[0]           ;GET RESULT
U 214F, B61E,95     ;25467          MOV LS[T15] TO WR[1]            ;EXPECTED DATA
U 2150, 0869,3C     ;25468          JSR [CHECK.RESULT]              ;SEE IF ARRAY WRITTEN CORRECTLY
U 2151, 0A13,B4     ;25469          JMP [LOOP.T43.1.2]              ;ERROR LOOP IF ENABLED
                    ;25470
U 2152, 3612,15     ;25471          MOV LS[T9] TO WR[0]             ;GET CURRENT ADDRESS
U 2153, C044,15     ;25472          ADD LS[#4] TO WR[0]             ;INCREMENT ADDRESS BY 4
```

```
U 2154, BE12,15 ;25473          MOV WR[0] TO LS[T9]                ;UPDATE ADDRESS IN LS
U 2155, BE88,15 ;25474          MOV WR[0] TO LS[ADDRESS.DATA]      ;UPDATE ADDRESS TO PRINT IF ERROR
                ;25475          CMP LS[#200] WITH WR[0],           ;SEE IF DONE 1 PAGE
U 2156, 4E52,35 ;25476           DT(LONG)&SET.ALU.CC               ; AND SET CONDITION CODES
U 2157, 8A14,D1 ;25477          JMP.IF[NEQ] TO [REPEAT.T43.READ] ;REPEAT UNTIL READ 1 PAGE
                ;25478
U 2158, 99BC,35 ;25479          MEM.REQ[WRITE.P] ADRS[CLEAR.ERR.ADDR] DT[WORD]
                ;25480                                             ;SET UP TO CLEAR ANY ERROR IN UBE ERROR
                ;25481                                             ; REG
U 2159, B29C,15 ;25482          WRITE.MEM LS[ZERO]                 ;CLEAR UBE ERROR REG
U 215A, 5B00,14 ;25483          RETURN                             ;DONE WITH NPR TEST FOR THIS DATA
                ;25484
                ;25485  END.T43:
U 215B, 0A17,94 ;25486          JMP [END.SECTION]                  ;DONE WITH NON-OPTIONAL TESTS
```

```
                    ;25487  .PAGE   " Read UBE Regs Routine (Debug routine.  Only run on DI TE 44)"
                    ;25488  ;++
                    ;25489  ;
                    ;25490  ;         This routine permits the operator to read the UBE registers to aid in
                    ;25491  ;       debug if necessary.  It will print out an error message for each regis-
                    ;25492  ;       ter with the data contained in that register under REC in the error
                    ;25493  ;       report.  All registers can be printed by clearing the halt on error
                    ;25494  ;       flag (CL HA) or individually by continuing after each report.
                    ;25495  ;
                    ;25496  ;         The test is executed only by typing DI TE 44 from the monitor.
                    ;25497  ;
                    ;25498  ;       Error 1 - Contents of UBE Data buffer.
                    ;25499  ;       Error 2 - Contents of UBE Cycle Count.
                    ;25500  ;       Error 3 - Contents of UBE Address buffer.
                    ;25501  ;       Error 4 - Contents of UBE Control Reg 1.
                    ;25502  ;       Error 5 - Contents of UBE Control Reg 2.
                    ;25503  ;
                    ;25504  ;--
                    ;25505
                    ;25506  READ.UBE.REGS:
U 215C, B65E,15     ;25507          MOV LS[BEGIN.TEST] TO WR[0]     ;SET BIT 15 IN WR[0] FOR CONTROL AND
                    ;25508                                          ; STATUS WORD
U 215D, 3E80,15     ;25509          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BIT 15 IN CONTROL AND STATUS WORD
                    ;25510                                          ; BIT 15 INDICATES START OF TEST TO
                    ;25511                                          ; CONSOLE
U 215E, 10E0,15     ;25512          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;25513  WAIT.READ.UBE.0:
U 215F, 8A15,F4     ;25514          JMP [WAIT.READ.UBE.0]           ;LOOP HERE WAITING FOR CONSOLE TO
                    ;25515                                          ; RESPOND
                    ;25516
U 2160, 0A17,EC     ;25517          JSR [CLEAR.CSR1]                ;CLEAR CSR 1 BITS
U 2161, 5F28,15     ;25518          MCOM LS[#FFFF] TO WR[0]
U 2162, 3E8A,15     ;25519          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK 0-15
U 2163, B640,15     ;25520          MOV LS[#1] TO WR[0]             ;1 IN WR
U 2164, BE82,15     ;25521          MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
                    ;25522
U 2165, 36B4,15     ;25523          MOV LS[BEDB] TO WR[0]           ;ADDRESS OF DATA BUFFER
U 2166, 8A17,0C     ;25524          JSR [READ.REG]                  ;READ REGISTER
U 2167, B6B6,15     ;25525          MOV LS[BECC] TO WR[0]           ;ADDRESS OF CYCLE COUNT
U 2168, 8A17,0C     ;25526          JSR [READ.REG]                  ;READ REGISTER
U 2169, 36B8,15     ;25527          MOV LS[BEBA] TO WR[0]           ;ADDRESS OF ADDR BUFFER
U 216A, 8A17,0C     ;25528          JSR [READ.REG]                  ;READ REGISTER
U 216B, B6BA,15     ;25529          MOV LS[BECR1] TO WR[0]          ;ADDRESS OF CONTROL REG 1
U 216C, 8A17,0C     ;25530          JSR [READ.REG]                  ;READ REGISTER
U 216D, 36BE,15     ;25531          MOV LS[BECR2] TO WR[0]          ;ADDRESS OF CONTROL REG 2
U 216E, 8A17,0C     ;25532          JSR [READ.REG]                  ;READ REGISTER
U 216F, 0A17,94     ;25533          JMP [END.SECTION]               ;DONE
                    ;25534
                    ;25535  READ.REG:
U 2170, BE12,15     ;25536          MOV WR[0] TO LS[T9]              ;ADDRESS IN LS
U 2171, 9813,B5     ;25537          MEM.REQ[READ.P] ADRS[T9] DT[WORD]
U 2172, 3022,15     ;25538          MOV MEM.DATA TO WR[0]            ;GET REGISTER CONTENTS
U 2173, 2000,95     ;25539          MOV WR[0] TO WR[1]
U 2174, A0C2,95     ;25540          COM WR[1]                        ;EXPECTED DATA IS COMP OF REC
U 2175, 0869,3C     ;25541          JSR [CHECK.RESULT]               ;FORCE ERROR PRINTOUT
```

```
U 2176, DB00,15 ;25542          NOP
U 2177, FF82,15 ;25543          INC LS[ERROR.NUMBER]            ;NEXT ERROR NUMBER
U 2178, 5B00,14 ;25544          RETURN                          ;DONE
                ;25545
```

```
                ;25546  .PAGE
                ;25547  END.SECTION:
U 2179, 365C,15 ;25548          MOV LS[EOS] TO WR[0]            ;SET UP WR 0 FOR END OF SECTION (BIT 14)
U 217A, 3E80,15 ;25549          MOV WR[0] TO LS[CONTROL.STATUS] ;SET UP CONTROL AND STATUS WORD TO
                ;25550                                          ; REPORT END OF SECTION
                ;25551
U 217B, 10E0,15 ;25552          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;25553  WAIT.EOS:
U 217C, 0A17,C4 ;25554          JMP [WAIT.EOS]                  ;LOOP HERE WAITING FOR CONSOLE TO
                ;25555                                          ; RESPOND
```

```
                    ;25556  .PAGE   'MCT SUBROUTINES'
                    ;25557  ;
                    ;25558  ;       COMMON SUBROUTINES FOR MCT TESTS
                    ;25559  ;
                    ;25560  WRITE.CSR1.MME:
                    ;25561          MOV LS[MME] TO WR[0],           ;SET MME BIT
U 217D, 3676,1E     ;25562           SKIP                           ; AND SKIP NEXT INSTRUCTION
                    ;25563  CLEAR.CSR1:
U 217E, 2F80,15     ;25564          CLR WR[0]                       ;PUT A ZERO IN WR
                    ;25565  WRITE.CSR1:
U 217F, BE1E,15     ;25566          MOV WR[0] TO LS[T15]            ;LOAD TEMPORARY LS LOCATION WITH DATA
                    ;25567                                          ; PATTERN
U 2180, 1D45,F5     ;25568          MEM.REQ[WRITE.CSR] ADRS[CSR1] DT[LONG]
                    ;25569                                          ;START WRITE TO CSR1
                    ;25570          WRITE.MEM LS[T15],              ;WRITE DATA PATTERN FROM LS LOCATION F(H)
U 2181, 321E,14     ;25571           RETURN                         ; RETURN TO MAIN PROGRAM
                    ;25572
                    ;25573  NEXT.TB.ADDRESS:
U 2182, 3612,15     ;25574          MOV LS[T9] TO WR[0]             ;GET OLD ADDRESS
U 2183, FF88,15     ;25575          INC LS[ADDRESS.DATA]            ;INCREMENT ADDRESS THAT GETS PRINTED
                    ;25576          ADD LS[BIT9] TO WR[0],          ;INCREMENT OLD ADDRESS
U 2184, C052,55     ;25577           DT(SIZE)&SET.ALU.CC            ; AND SET CONDITION CODES
U 2185, 8A18,A0     ;25578          JMP.IF[N.CLR] TO [RET.NEXT]     ;REPEAT IF BIT 15 IS STILL CLEAR
U 2186, 455E,15     ;25579          BIC LS[BIT15] TO WR[0]          ;ELSE CLEAR BIT 15
                    ;25580          XOR LS[BIT31] TO WR[0],         ;TOGGLE BIT 31
U 2187, 437E,35     ;25581           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 2188, DB00,08     ;25582          SKIP.IF[N.SET]                  ;SKIP NEXT INSTRUCTION IF BIT 31 IS SET
                    ;25583                                          ; ELSE WE ARE DONE
U 2189, DB00,16     ;25584          RETURN+1                        ;GO TO NEXT PART
                    ;25585  RET.NEXT:
U 218A, BE12,15     ;25586          MOV WR[0] TO LS[T9]             ;STORE NEW ADDRESS IN LS 9
U 218B, 5B00,14     ;25587          RETURN                          ;REPEAT TEST AT NEW ADDRESS
                    ;25588
                    ;25589  NEXT.UB.ADDRESS:
U 218C, 3612,15     ;25590          MOV LS[T9] TO WR[0]             ;GET OLD ADDRESS
U 218D, FF88,15     ;25591          INC LS[ADDRESS.DATA]            ;INCREMENT ADDRESS THAT GETS PRINTED
U 218E, 4052,15     ;25592          ADD LS[BIT9] TO WR[0]           ;INCREMENT OLD ADDRESS
                    ;25593          BIT LS[BIT18] WITH WR[0],       ;SEE IF BIT 18 HAS SET
U 218F, 5964,35     ;25594           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 2190, 5B00,09     ;25595          SKIP.IF[BITS.CLR]               ;REPEAT IF BIT 18 IS STILL CLEAR
                    ;25596                                          ; ELSE WE ARE DONE
U 2191, DB00,16     ;25597          RETURN+1                        ;GO TO NEXT PART
                    ;25598
U 2192, BE12,15     ;25599          MOV WR[0] TO LS[T9]             ;STORE NEW ADDRESS IN LS 9
U 2193, 5B00,14     ;25600          RETURN                          ;REPEAT TEST AT NEW ADDRESS
                    ;25601
                    ;25602  ;
                    ;25603  ; This routine sizes and initializes main memory and prints the number of 256K
                    ;25604  ; blocks found.
                    ;25605  ;
                    ;25606
                    ;25607  SIZE.MEMORY:
U 2194, 6524,15     ;25608          CLR LS[MM.LASTADDR+1]           ;CLEAR AN LS LOCATION (LOCATION 12(H))
                    ;25609                                          ; FOR STORAGE OF ADDRESS
U 2195, B645,15     ;25610          MOV LS[#4] TO WR[2]             ;INCREMENT VALUE FOR MEM ADDRESS
```

```
U 2196, B640,15 ;25611          MOV LS[#1] TO WR[0]             ;1 IN WR
                ;25612  REPEAT.SIZE.1:
U 2197, 9924,75 ;25613          MEM.REQ[WRITE.P] ADRS[MM.LASTADDR+1] DT[LONG]
                ;25614                                          ;SET UP TO WRITE 0'S THROUGHOUT MEMORY
U 2198, B29C,15 ;25615          WRITE.MEM LS[ZERO]              ;WRITE ALL ZEROS IN MEMORY
U 2199, 5B00,1D ;25616          SKIP.IF[MEM.REF.OK]             ;SKIP NEXT INTRUCTION IF NO ERROR SUM
U 219A, 0A19,D4 ;25617          JMP [PRINT.SIZE]                ;JMP TO PRINT MEMORY SIZE CALCULATED
U 219B, EC25,15 ;25618          ADD WR[2] TO LS[MM.LASTADDR+1]  ;INCREMENT MEMORY ADDRESS BY 4
U 219C, 0A19,74 ;25619          JMP [REPEAT.SIZE.1]             ;REPEAT UNTIL GET ERROR SUM
                ;25620
                ;25621  PRINT.SIZE:
U 219D, B625,15 ;25622          MOV LS[MM.LASTADDR+1] TO WR[2]  ;PUT LASTADDR+1 IN WR
U 219E, 3648,15 ;25623          MOV LS[#10] TO WR[0]            ;16 DECIMAL IN WR 0
U 219F, C042,15 ;25624          ADD LS[#2] TO WR[0]             ;18 DECIMAL IN WR 0
                ;25625  SIZE.SHIFT.LOOP:
U 21A0, A341,15 ;25626          ROR WR[2]                       ;SHIFT LAST ADDRESS+1 RIGHT
                ;25627          DEC WR[0],                      ;DECREMENT COUNTER
U 21A1, A100,35 ;25628           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 21A2, 8A1A,01 ;25629          JMP.IF[NEQ] TO [SIZE.SHIFT.LOOP];REPEAT UNTIL SHIFTED RIGHT 18 PLACES
                ;25630
U 21A3, BE0F,15 ;25631          MOV WR[2] TO LS[MEMORY.SIZE]    ;PUT SIZE IN 256KB BLOCKS IN LS 7 FOR
                ;25632                                          ; PRINTING
U 21A4, 3656,15 ;25633          MOV LS[PRINT.MEM.SIZE] TO WR[0] ;SET BIT 11 IN WR
U 21A5, 3E80,15 ;25634          MOV WR[0] TO LS[CONTROL.STATUS] ;INFORMS CONSOLE TO PRINT MEMORY SIZE
U 21A6, 10E0,15 ;25635          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                ;25636  WAIT.SIZE.1:
U 21A7, 0A1A,74 ;25637          JMP [WAIT.SIZE.1]               ;LOOP HERE WAITING FOR CONSOLE TO
                ;25638                                          ; RESPOND
                ;25639          MOV LS[MM.LASTADDR+1] TO WR[2], ;STORE SIZE IN WR 2
U 21A8, 3625,14 ;25640           RETURN                         ; DONE
                ;25641
                ;25642  ;
                ;25643  ; This routine performs the setup at the beginning of each test.  SETUP.1
                ;25644  ; does not clear CSR 1, while SETUP does.
                ;25645  ;
                ;25646
U 21A9, 0A17,EC ;25647  SETUP:  JSR [CLEAR.CSR1]                ;CLEAR MCT CSR 1
                ;25648
                ;25649  SETUP.1:
U 21AA, 65A0,15 ;25650          CLR LS[PREVIOUS.ERROR]          ;CLEAR PREVIOUS ERROR NUMBER IN LS
U 21AB, E58A,15 ;25651          CLR LS[ERROR.MASK]              ;SET UP ERROR MASK TO CHECK ALL BITS
U 21AC, 3022,15 ;25652          MOV MEM.DATA TO WR[0]           ;USED TO FREE MEMORY IF WAITING FOR A
                ;25653                                          ; CPU DR
U 21AD, 3022,15 ;25654          MOV MEM.DATA TO WR[0]           ;NEED 4 OF THESE IN CASE HUNG IN OCTA
U 21AE, 3022,15 ;25655          MOV MEM.DATA TO WR[0]
U 21AF, 3022,15 ;25656          MOV MEM.DATA TO WR[0]
U 21B0, B640,15 ;25657          MOV LS[#1] TO WR[0]             ;1 IN WR
U 21B1, BE82,15 ;25658          MOV WR[0] TO LS[ERROR.NUMBER]   ;SET UP ERROR NUMBER IN LS
U 21B2, 3644,15 ;25659          MOV LS[MCT] TO WR[0]            ;BIT 2 SET
                ;25660          MOV WR[0] TO LS[MODULE.NUM],    ;STORE MODULE CODE IN LS TO INDICATE
                ;25661                                          ; MCT BOARD
U 21B3, BE8C,14 ;25662           RETURN                         ;DONE WITH SETUP
                ;25663
                ;25664  ;
                ;25665  ; This routine performs a 4 CPU cycle delay
```

```
                    ;25666  ;
                    ;25667
                    ;25668  NOP.DELAY:
U 21B4, DB00,15     ;25669          NOP                             ;DELAY 1 CYCLE
U 21B5, DB00,15     ;25670          NOP                             ;AGAIN
U 21B6, DB00,15     ;25671          NOP                             ;AGAIN
U 21B7, 5B00,14     ;25672          RETURN                          ;LAST DELAY CYCLE
                    ;25673
                    ;25674  ;
                    ;25675  ;  Common Subroutines for UBE Tests
                    ;25676  ;
                    ;25677
                    ;25678  ISSUE.DCLO:
U 21B8, 3680,95     ;25679          MOV LS[CONTROL.STATUS] TO WR[1] ;SAVE OLD CONTROL AND STATUS WORD
U 21B9, 3660,15     ;25680          MOV LS[INTERUPT.EN] TO WR[0]    ;SET BIT 16 IN WR
U 21BA, C77A,15     ;25681          BIS LS[UBS.DCLO] TO WR[0]       ;ALSO SET BIT 29
U 21BB, 3E80,15     ;25682          MOV WR[0] TO LS[CONTROL.STATUS] ;SET BITS IN CONTROL AND STATUS WORD TO
                    ;25683                                          ; TELL MONITOR TO ASSERT THEN DEASSERT
                    ;25684                                          ; UNIBUS DCLO
U 21BC, 10E0,15     ;25685          MISC [SET.CP.ATTN]              ;ISSUE CPU ATTN TO CONSOLE
                    ;25686  WAIT.ISSUE.0:
U 21BD, 8A1B,D4     ;25687          JMP [WAIT.ISSUE.0]              ;LOOP HERE WAITING FOR CONSOLE TO
U 21BE, BE80,95     ;25688          MOV WR[1] TO LS[CONTROL.STATUS] ;RESTORE OLD CONTROL AND STATUS WORD
U 21BF, 5B00,14     ;25689          RETURN                          ;RETURN TO MAIN PROGRAM
                    ;25690
                    ;25691
                    ;25692  SETUP.DATO.NPR:
U 21C0, B646,15     ;25693          MOV LS[BR6] TO WR[0]            ;SET BIT 3 IN WR (INTR AT BR6)
U 21C1, C74C,15     ;25694          BIS LS[INT.ODNE] TO WR[0]       ;SET BIT 6 IN WR (INTERRUPT ON DONE)
U 21C2, C74A,15     ;25695          BIS LS[NPR] TO WR[0]            ;SET BIT 5 IN WR (ISSUE NPR)
U 21C3, C752,15     ;25696          BIS LS[C01] TO WR[0]            ;SET BIT 9 IN WR (EXECUTE DATO CYCLE)
U 21C4, C754,15     ;25697          BIS LS[FUNA] TO WR[0]           ;SET BIT 10 IN WR (EXECUTE 1 CYCLE 'TIL
                    ;25698                                          ; CYCLE COUNT OVERFLOW)
U 21C5, C740,15     ;25699          BIS LS[GO] TO WR[0]             ;SET BIT 0 (UBE GO BIT)
                    ;25700          MOV WR[0] TO LS[T14],           ;SAVE IN LS T14
U 21C6, BE1C,14     ;25701           RETURN                         ; DONE
                    ;25702
                    ;25703
                    ;25704  SETUP.DATI.NPR:
U 21C7, B646,15     ;25705          MOV LS[BR6] TO WR[0]            ;SET BIT 3 IN WR (INTR AT BR6)
U 21C8, C74C,15     ;25706          BIS LS[INT.ODNE] TO WR[0]       ;SET BIT 6 IN WR (INTERRUPT ON DONE)
U 21C9, C74A,15     ;25707          BIS LS[NPR] TO WR[0]            ;SET BIT 5 IN WR (ISSUE NPR)
U 21CA, C754,15     ;25708          BIS LS[FUNA] TO WR[0]           ;SET BIT 10 IN WR (EXECUTE 1 CYCLE 'TIL
                    ;25709                                          ; CYCLE COUNT OVERFLOW)
U 21CB, C740,15     ;25710          BIS LS[GO] TO WR[0]             ;SET BIT 0 (UBE GO BIT)
                    ;25711          MOV WR[0] TO LS[T14],           ;SAVE IN LS T14
U 21CC, BE1C,14     ;25712           RETURN                         ; DONE
                    ;25713
                    ;25714
                    ;25715  CHECK.UBE.ERR:
U 21CD, 36FC,15     ;25716          MOV LS[INTERRUPT.VEC] TO WR[0]  ;GET BR INTERRUPT IDENTIFIER
U 21CE, B710,95     ;25717          MOV LS[IDENT.MASK] TO WR[1]     ;FFFFFFC3 IN WR 1
U 21CF, AF42,15     ;25718          BIC WR[1] TO WR[0]              ;CLEAR ALL BUT BITS 5-2 IN IDENTIFIER
                    ;25719          CMP LS[BR7.IDENT] WITH WR[0],   ;SEE IF BR7 (UBE ERROR)
U 21D0, CECC,35     ;25720           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
```

```
U 21D1, 8A1D,99 ;25721          JMP.IF[EQL] TO [UBE.ERROR]      ;GO ISSUE BG 7 AND REPORT ERROR IF BR7
U 21D2, 9F46,35 ;25722          MEM.REQ[ISSUE.BG] ADRS[BG6] DT[WORD]
                ;25723                                          ;SET UP TO ISSUE BG ON LEVEL 6
U 21D3, 3022,15 ;25724          MOV MEM.DATA TO WR[0]           ;COMPLETE BG CYCLE
U 21D4, 3642,15 ;25725          MOV LS[#2] TO WR[0]             ;DELAY COUNT
                ;25726  LOOP.NOP:
U 21D5, DB00,15 ;25727          NOP                             ;ALLOW TIME FOR BR7 INTERRUPT IF UBE
                ;25728                                          ; ERROR
                ;25729          DEC WR[0],                      ;DECREMENT COUNT
U 21D6, A100,35 ;25730           DT(LONG)&SET.ALU.CC            ; AND SET CONDITION CODES
U 21D7, 0A1D,51 ;25731          JMP.IF[NEQ] TO [LOOP.NOP]       ;REPEAT TIL COUNT EXHAUSTED
                ;25732
U 21D8, 8A1E,27 ;25733          JMP.IF[NO.INTERRUPT] TO [CHECK.UBE.ERROR.RET+1]
                ;25734                                          ;RETURN IF NO BR7 PENDING
                ;25735                                          ; ELSE REPORT UBE ERROR AND UBE ERROR
                ;25736                                          ; REG CONTENTS
                ;25737  UBE.ERROR:
U 21D9, 9FCE,35 ;25738          MEM.REQ[ISSUE.BG] ADRS[BG7] DT[WORD]
                ;25739                                          ;SET UP TO ISSUE BG AT LEVEL 7
U 21DA, 3022,15 ;25740          MOV MEM.DATA TO WR[0]           ;COMPLETE BG CYCLE
U 21DB, 5F28,15 ;25741          MCOM LS[#FFFF] TO WR[0]         ;FFFF0000 IN WR
U 21DC, 3E8A,15 ;25742          MOV WR[0] TO LS[ERROR.MASK]     ;CHECK BITS 15-00 IN UBE CONTROL REG 2
U 21DD, 98BF,B5 ;25743          MEM.REQ[READ.P] ADRS[BECR2] DT[WORD]
                ;25744                                          ;READ UBE CONTROL REG 2
U 21DE, 3022,15 ;25745          MOV MEM.DATA TO WR[0]           ;GET CONTENTS OF CONTROL REG 2
U 21DF, 2F82,95 ;25746          CLR WR[1]                       ;EXPECTED DATA (ERROR BITS CLEAR)
U 21E0, 0869,3C ;25747          JSR [CHECK.RESULT]              ;REPORT UBE ERROR
U 21E1, 5B00,14 ;25748          RETURN                          ;ERROR LOOP RETURN
                ;25749  CHECK.UBE.ERROR.RET+1:
U 21E2, DB00,16 ;25750          RETURN+1                        ;NO LOOP ON ERROR RETURN
                ;25751
                ;25752  CLEAR.CSR2:
U 21E3, 9D47,F5 ;25753          MEM.REQ[WRITE.CSR] ADRS[CSR2] DT[LONG]
                ;25754                                          ;SET UP TO WRITE TO CSR 2
                ;25755          WRITE.MEM LS[ZERO],             ;CLEAR ERRORS IN UB CSR 2
U 21E4, 329C,14 ;25756           RETURN                         ;DONE
                ;25757
```

Cross Reference Listing - Field Names and Defined Values

Cross Reference Listing - Macro Names

Cross Reference Listing - Expression Names

;
; Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
C 0000	1783:	1784:	1785:	1786:	1787:	1788:	1789:	1790:
C 0008	1791:	1792:	1793:	1794:	1795:	1796:	1797:	1798:
C 0010	1802:	1803:	1804:	1805:	1806:	1807:	1808:	1809:
C 0018	1810:	1811:	1812:	1813:	1814:	1815:	1816:	1817:
C 0020	1826:	1827:	1828:	1829:	1830:	1831:	1832:	1833:
C 0028	1834:	1835:	1836:	1837:	1838:	1839:	1840:	1841:
C 0030	1842:	1843:	1844:	1845:	1846:	1847:	1848:	1849:
C 0038	1850:	1851:	1852:	1853:	1854:	1855:	1856:	1857:
C 0040	1866:	1867:	1868:	1869:	1870:	1871:	1872:	1873:
C 0048	1874:	1875:	1876:	1877:	1878:	1879:	1880:	1881:
C 0050	1882:	1883:	1884:	1885:	1886:	1887:	1888:	1889:
C 0058	1890:	1891:	1892:	1893:	1894:	1895:	1896:	1897:
C 0060	1905:	1906:	1907:	1908:	1909:	1910:	1911:	1912:
C 0068	1913:	1914:	1915:	1916:	1917:	1918:	1919:	1920:
C 0070	1921:	1922:	1923:	1924:	1925:	1926:	1927:	1928:
C 0078	1929:	1930:	1931:	1932:	1933:	1934:	1935:	1936:
C 0080	1943:	1945:	1947:	1949:	1952:	1954:	1956:	1958:
C 0088	1961:	1963:	1965:	1967:	1970:	1972:	1974:	1976:
C 0090	1979:	1981:	1983:	1985:	1988:	1990:	1992:	1994:
C 0098	1997:	1999:	2001:	2003:	2006:	2008:	2010:	2012:
C 00A0	2015:	2017:	2019:	2021:	2024:	2026:	2028:	2030:
C 00A8	2033:	2035:	2037:	2039:	2042:	2044:	2046:	2048:
C 00B0	2051:	2053:	2055:	2057:	2060:	2062:	2064:	2066:
C 00B8	2069:	2071:	2073:	2075:	2078:	2080:	2082:	2084:
C 00C0	2092:	2093:	2094:	2095:	2096:	2097:	2098:	2099:
C 00C8	2102:	2103:	2104:	2105:	2106:	2107:	2108:	2109:
C 00D0	2112:	2113:	2114:	2115:	2116:	2117:	2118:	2119:
C 00D8	2122:	2123:	2124:	2125:	2126:	2127:	2128:	2129:
C 00E0	2133:	2134:	2135:	2136:	2137:	2138:	2139:	2140:
C 00E8	2142:	2143:	2144:	2145:	2146:	2147:	2148:	2149:
C 00F0	2153:	2154:	2155:	2156:	2157:	2158:	2159:	2160:
C 00F8	2163:	2164:	2165:	2166:	2167:	2168:	2169:	2170:
C 0100	2178:	2179:	2180:	2181:	2183:	2184:	2185:	2186:
C 0108	2188:	2189:	2190:	2191:	2193:	2194:	2195:	2196:
C 0110	2199:	2200:	2201:	2202:	2204:	2205:	2206:	2207:
C 0118	2209:	2210:	2211:	2212:	2214:	2215:	2216:	2217:
C 0120	2220:	2221:	2222:	2223:	2225:	2226:	2227:	2228:
C 0128	2230:	2231:	2232:	2233:	2235:	2236:	2237:	2238:
C 0130	2241:	2242:	2243:	2244:	2246:	2247:	2248:	2249:
C 0138	2251:	2252:	2253:	2254:	2256:	2257:	2258:	2259:
C 0140	2262:	2263:	2264:	2265:	2267:	2268:	2269:	2270:
C 0148	2272:	2273:	2274:	2275:	2277:	2278:	2279:	2280:
C 0150	2283:	2284:	2285:	2286:	2288:	2289:	2290:	2291:
C 0158	2293:	2294:	2295:	2296:	2298:	2299:	2300:	2301:
C 0160	2304:	2305:	2306:	2307:	2309:	2310:	2311:	2312:
C 0168	2314:	2315:	2316:	2317:	2319:	2320:	2321:	2322:
C 0170	2325:	2326:	2327:	2328:	2330:	2331:	2332:	2333:
C 0178	2335:	2336:	2337:	2338:	2340:	2341:	2342:	2343:
C 0180	2353:	2354:	2355:	2356:	2358:	2359:	2360:	2361:
C 0188	2363:	2364:	2365:	2366:	2368:	2369:	2370:	2371:
C 0190	2374:	2375:	2376:	2377:	2379:	2380:	2381:	2382:
C 0198	2384:	2385:	2386:	2387:	2389:	2390:	2391:	2392:
C 01A0	2395:	2396:	2397:	2398:	2400:	2401:	2402:	2403:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
C 01A8	2405:	2406:	2407:	2408:	2410:	2411:	2412:	2413:
C 01B0	2416:	2417:	2418:	2419:	2421:	2422:	2423:	2424:
C 01B8	2426:	2427:	2428:	2429:	2431:	2432:	2433:	2434:
C 01C0	2437:	2438:	2439:	2440:	2442:	2443:	2444:	2445:
C 01C8	2447:	2448:	2449:	2450:	2452:	2453:	2454:	2455:
C 01D0	2458:	2459:	2460:	2461:	2463:	2464:	2465:	2466:
C 01D8	2468:	2469:	2470:	2471:	2473:	2474:	2475:	2476:
C 01E0	2479:	2480:	2481:	2482:	2484:	2485:	2486:	2487:
C 01E8	2489:	2490:	2491:	2492:	2494:	2495:	2496:	2497:
C 01F0	2500:	2501:	2502:	2503:	2505:	2506:	2507:	2508:
C 01F8	2510:	2511:	2512:	2513:	2515:	2516:	2517:	2518:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 0000	4024:	3912:	3913:	3914:	3915:	3916:	3917:	3918:
U 0008	3919:	3920:	3921:	3922:	3923:	3924:	3925:	3926:
U 0010	3927:	3928:	3929:	3930:	3931:	3932:	3933:	3934:
U 0018	3935:	3936:	3937:	3938:	3939:	3940:	3941:	3942:
U 0020	3943:	3944:	3945:	3946:	3947:	3948:	3949:	3950:
U 0028	3951:	3952:	3953:	3954:	3955:	3956:	3957:	3958:
U 0030	3959:	3960:	3961:	3962:	3963:	3964:	3965:	3966:
U 0038	3967:	3968:	3969:	3970:	3971:	3972:	3973:	3974:
U 0040	3975:	3976:	3977:	3978:	3979:	3981:	3983:	3985:
U 0048	3987:	3989:	3991:	3993:	3995:	3997:	3999:	4001:
U 0050	4033:	4035:	4037:	4040:	4043:	4047:	4049:	4051:
U 0058	4053:	4055:	4057:	4059:	4061:	4063:	4065:	4067:
U 0060	4069:	4071:	4073:	4075:	4077:	4079:	4080:	4081:
U 0068	4082:	4083:	4084:	4085:	4086:	4087:	4088:	4089:
U 0070	4119:	4121:	4123:	4124:	4126:	4127:	4129:	4130:
U 0078	4132:	4133:	4135:	4136:	4138:	4139:	4141:	4142:
U 0080	4144:	4145:	4147:	4148:	4150:	4151:	4153:	4154:
U 0088	4156:	4157:	4159:	4160:	4162:	4163:	4165:	4166:
U 0090	4168:	4169:	4171:	4172:	4174:	4175:	4177:	4178:
U 0098	4180:	4181:	4183:	4184:	4186:	4187:	4189:	4190:
U 00A0	4192:	4193:	4195:	4196:	4198:	4199:	4201:	4202:
U 00A8	4204:	4205:	4207:	4208:	4210:	4211:	4213:	4214:
U 00B0	4216:	4217:	4219:	4220:	4222:	4223:	4225:	4226:
U 00B8	4228:	4229:	4231:	4232:	4234:	4235:	4237:	4238:
U 00C0	4240:	4241:	4243:	4244:	4246:	4247:	4249:	4250:
U 00C8	4252:	4253:	4255:	4256:	4258:	4259:	4261:	4262:
U 00D0	4264:	4265:	4267:	4268:	4270:	4271:	4273:	4274:
U 00D8	4276:	4277:	4279:	4280:	4282:	4283:	4285:	4286:
U 00E0	4288:	4289:	4291:	4292:	4294:	4295:	4297:	4298:
U 00E8	4300:	4301:	4303:	4304:	4306:	4307:	4309:	4310:
U 00F0	4312:	4313:	4315:	4316:	4318:	4319:	4321:	4322:
U 00F8	4324:	4325:	4327:	4328:	4330:	4331:	4333:	4334:
U 0100	4336:	4337:	4339:	4340:	4342:	4343:	4345:	4346:
U 0108	4348:	4349:	4351:	4352:	4354:	4355:	4357:	4358:
U 0110	4360:	4361:	4363:	4364:	4366:	4367:	4369:	4370:
U 0118	4372:	4373:	4375:	4376:	4378:	4379:	4381:	4382:
U 0120	4384:	4385:	4387:	4388:	4390:	4391:	4393:	4394:
U 0128	4396:	4397:	4399:	4400:	4402:	4403:	4405:	4406:
U 0130	4408:	4409:	4411:	4412:	4414:	4415:	4417:	4418:
U 0138	4420:	4421:	4423:	4424:	4426:	4427:	4429:	4430:
U 0140	4432:	4433:	4435:	4436:	4438:	4439:	4441:	4442:
U 0148	4444:	4445:	4447:	4448:	4450:	4451:	4453:	4454:
U 0150	4456:	4457:	4459:	4460:	4462:	4463:	4465:	4466:
U 0158	4468:	4469:	4471:	4472:	4474:	4475:	4477:	4478:
U 0160	4480:	4481:	4490:	4492:	4494:	4495:	4497:	4498:
U 0168	4500:	4501:	4503:	4504:	4506:	4507:	4509:	4510:
U 0170	4512:	4513:	4515:	4516:	4518:	4519:	4521:	4522:
U 0178	4524:	4525:	4527:	4528:	4530:	4531:	4533:	4534:
U 0180	4536:	4537:	4539:	4540:	4542:	4543:	4545:	4546:
U 0188	4548:	4549:	4551:	4552:	4554:	4555:	4557:	4558:
U 0190	4560:	4561:	4563:	4564:	4566:	4567:	4569:	4570:
U 0198	4572:	4573:	4575:	4576:	4578:	4579:	4581:	4582:
U 01A0	4584:	4585:	4587:	4588:	4590:	4591:	4593:	4594:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 01A8	4596:	4597:	4599:	4600:	4602:	4603:	4605:	4606:
U 01B0	4608:	4609:	4611:	4612:	4614:	4615:	4617:	4618:
U 01B8	4620:	4621:	4623:	4624:	4626:	4627:	4629:	4630:
U 01C0	4632:	4633:	4635:	4636:	4638:	4639:	4641:	4642:
U 01C8	4644:	4645:	4647:	4648:	4650:	4651:	4653:	4654:
U 01D0	4656:	4657:	4659:	4660:	4662:	4663:	4665:	4666:
U 01D8	4668:	4669:	4671:	4672:	4674:	4675:	4677:	4678:
U 01E0	4680:	4681:	4683:	4684:	4686:	4687:	4689:	4690:
U 01E8	4692:	4693:	4695:	4696:	4698:	4699:	4701:	4702:
U 01F0	4704:	4705:	4707:	4708:	4710:	4711:	4713:	4714:
U 01F8	4716:	4717:	4719:	4720:	4722:	4723:	4725:	4726:
U 0200	4728:	4729:	4731:	4732:	4734:	4735:	4737:	4738:
U 0208	4740:	4741:	4743:	4744:	4746:	4747:	4749:	4750:
U 0210	4752:	4753:	4755:	4756:	4758:	4759:	4761:	4762:
U 0218	4764:	4765:	4767:	4768:	4770:	4771:	4773:	4774:
U 0220	4776:	4777:	4779:	4780:	4782:	4783:	4785:	4786:
U 0228	4788:	4789:	4791:	4792:	4794:	4795:	4797:	4798:
U 0230	4800:	4801:	4803:	4804:	4806:	4807:	4809:	4810:
U 0238	4812:	4813:	4815:	4816:	4818:	4819:	4821:	4822:
U 0240	4824:	4825:	4827:	4828:	4830:	4831:	4833:	4834:
U 0248	4836:	4837:	4839:	4840:	4842:	4843:	4845:	4846:
U 0250	4848:	4849:	4851:	4852:				
U 0258 - 05FF Unused								
U 0600	4878:	4879:	4881:	4884:	4885:	4886:	4900:	4901:
U 0608	4962:	4964:	4966:	4968:	4969:	5024:	5025:	5027:
U 0610	5029:	5031:	5034:	5036:	5037:	5038:	5042:	5043:
U 0618	5044:	5048:	5049:	5050:	5052:	5055:	5056:	5111:
U 0620	5113:	5114:	5117:	5119:	5123:	5125:	5129:	5135:
U 0628	5137:	5138:	5144:	5146:	5148:	5149:	5150:	5156:
U 0630	5157:	5158:	5159:	5160:	5161:	5162:	5165:	5167:
U 0638	5168:	5172:	5222:	5224:	5226:	5228:	5231:	5232:
U 0640	5234:	5235:	5290:	5292:	5293:	5296:	5298:	5302:
U 0648	5304:	5308:	5314:	5316:	5317:	5323:	5324:	5325:
U 0650	5326:	5327:	5328:	5331:	5381:	5383:	5386:	5388:
U 0658	5391:	5392:	5394:	5395:	5443:	5448:	5450:	5452:
U 0660	5453:	5454:	5504:	5506:	5509:	5513:	5515:	5516:
U 0668	5555:	5556:	5557:	5560:	5561:	5562:	5565:	5566:
U 0670	5567:	5570:	5571:	5572:	5575:	5576:	5580:	5581:
U 0678	5582:	5621:	5622:	5623:	5627:	5628:	5629:	5633:
U 0680	5634:	5635:	5639:	5640:	5644:	5645:	5649:	5652:
U 0688	5654:	5701:	5702:	5703:	5704:	5705:	5752:	5753:
U 0690	5754:	5755:	5756:	5824:	5826:	5829:	5830:	5832:
U 0698	5834:	5835:	5836:	5838:	5842:	5844:	5846:	5850:
U 06A0	5851:	5854:	5855:	5857:	5859:	5860:	5862:	5864:
U 06A8	5869:	5870:	5872:	5874:	5875:	5881:	5882:	5885:
U 06B0	5886:	5887:	5892:	5893:	5895:	5896:	5898:	5900:
U 06B8	5901:	5903:	5905:	5907:	5909:	5911:	5918:	5919:
U 06C0	5921:	5924:	5937:	5938:	5940:	5941:	5942:	5943:
U 06C8	5944:	5945:	5946:	5947:	5948:	5952:	5955:	5956:
U 06D0	5957:	5958:	5959:	5960:	5961:	5962:	5963:	5964:
U 06D8	5965:	5966:	5967:	5968:	5969:	5970:	5973:	5976:
U 06E0	5979:	5981:	5982:	5983:	5984:	5987:	5990:	5993:
U 06E8	5994:	5995:	5997:	5998:	5999:	6000:	6002:	6003:

Location / Line Number Index

Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 06F0	6004:	6005:	6007:	6008:	6011:	6013:	6014:	
U 06F8 - 06FF Unused								
U 0700	6328:	6330:	6333:	6335:	6338:	6339:	6340:	6343:
U 0708	6345:	6347:	6348:	6349:	6351:	6353:	6354:	6356:
U 0710	6357:	6358:	6360:	6361:	6363:	6364:	6366:	6368:
U 0718	6369:	6370:	6372:	6373:	6375:	6376:	6378:	6379:
U 0720	6381:	6383:	6385:	6387:	6389:	6390:	6391:	6393:
U 0728	6394:	6396:	6397:	6399:	6401:	6403:	6405:	6407:
U 0730	6408:	6409:	6527:	6529:	6532:	6534:	6537:	6538:
U 0738	6539:	6542:	6543:	6545:	6547:	6549:	6550:	6551:
U 0740	6553:	6554:	6555:	6556:	6557:	6558:	6560:	6561:
U 0748	6563:	6565:	6568:	6569:	6570:	6572:	6573:	6575:
U 0750	6576:	6577:	6579:	6581:	6582:	6584:	6586:	6588:
U 0758	6589:	6590:	6592:	6593:	6595:	6596:	6598:	6599:
U 0760	6601:	6603:	6606:	6607:	6609:	6611:	6614:	6615:
U 0768	6616:	6619:	6620:	6621:	6623:	6625:	6626:	6627:
U 0770	6629:	6631:	6632:	6634:	6636:	6638:	6639:	6640:
U 0778	6642:	6643:	6644:	6647:	6648:	6650:	6652:	6656:
U 0780	6657:	6800:	6802:	6805:	6807:	6810:	6811:	6812:
U 0788	6814:	6815:	6816:	6817:	6818:	6819:	6820:	6822:
U 0790	6824:	6825:	6827:	6829:	6830:	6832:	6833:	6834:
U 0798	6836:	6837:	6839:	6840:	6842:	6844:	6845:	6847:
U 07A0	6848:	6849:	6851:	6852:	6855:	6856:	6858:	6859:
U 07A8	6861:	6862:	6863:	6865:	6866:	6868:	6869:	7024:
U 07B0	7026:	7029:	7031:	7034:	7035:	7036:	7039:	7040:
U 07B8	7042:	7043:	7044:	7046:	7047:	7048:	7049:	7051:
U 07C0	7052:	7055:	7056:	7057:	7059:	7060:	7061:	7063:
U 07C8	7064:	7067:	7068:	7069:	7071:	7072:	7073:	7075:
U 07D0	7076:	7077:	7080:	7081:	7082:	7084:	7085:	7087:
U 07D8	7088:	7091:	7092:	7093:	7198:	7200:	7203:	7205:
U 07E0	7208:	7210:	7211:	7213:	7215:	7216:	7219:	7220:
U 07E8	7221:	7223:	7224:	7226:	7228:	7229:	7232:	7233:
U 07F0	7234:	7236:	7237:	7239:	7240:	7241:	7243:	7245:
U 07F8	7247:	7248:	7251:	7252:	7253:	7255:	7256:	7258:
U 0800	7259:	7356:	7358:	7361:	7363:	7366:	7367:	7368:
U 0808	7370:	7371:	7374:	7375:	7378:	7379:	7381:	7382:
U 0810	7383:	7385:	7387:	7388:	7390:	7391:	7392:	7394:
U 0818	7395:	7397:	7398:	7400:	7401:	7403:	7404:	7405:
U 0820	7407:	7408:	7410:	7411:	7472:	7474:	7477:	7479:
U 0828	7482:	7483:	7485:	7488:	7489:	7490:	7492:	7493:
U 0830	7494:	7495:	7497:	7499:	7500:	7502:	7503:	7504:
U 0838	7505:	7704:	7706:	7709:	7711:	7714:	7715:	7716:
U 0840	7718:	7719:	7720:	7721:	7722:	7724:	7725:	7727:
U 0848	7728:	7729:	7730:	7732:	7733:	7735:	7736:	7740:
U 0850	7741:	7742:	7743:	7866:	7868:	7871:	7873:	7876:
U 0858	7877:	7878:	7880:	7881:	7883:	7884:	7885:	7886:
U 0860	7887:	7889:	7890:	7893:	7894:	7895:	8057:	8059:
U 0868	8062:	8064:	8067:	8070:	8071:	8073:	8074:	8076:
U 0870	8077:	8078:	8080:	8082:	8083:	8085:	8086:	8088:
U 0878	8089:	8090:	8092:	8093:	8095:	8096:	8098:	8099:
U 0880	8101:	8102:	8103:	8105:	8106:	8108:	8109:	8310:
U 0888	8312:	8315:	8317:	8320:	8321:	8324:	8326:	8327:
U 0890	8328:	8331:	8332:	8333:	8334:	8336:	8338:	8339:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 0898	8340:	8342:	8343:	8344:	8345:	8347:	8348:	8349:
U 08A0	8351:	8353:	8354:	8355:	8356:	8360:	8361:	8362:
U 08A8	8363:	8365:	8367:	8368:	8369:	8371:	8372:	8373:
U 08B0	8374:	8376:	8377:	8378:	8380:	8382:	8384:	8385:
U 08B8	8386:	8387:	8388:	8392:	8393:	8394:	8395:	8397:
U 08C0	8399:	8400:	8401:	8403:	8404:	8405:	8406:	8408:
U 08C8	8409:	8410:	8412:	8413:	8415:	8416:	8606:	8608:
U 08D0	8611:	8613:	8616:	8618:	8620:	8621:	8623:	8625:
U 08D8	8628:	8630:	8631:	8632:	8633:	8634:	8636:	8637:
U 08E0	8638:	8639:	8641:	8642:	8643:	8644:	8646:	8647:
U 08E8	8648:	8651:	8652:	8653:	8654:	8655:	8657:	8658:
U 08F0	8659:	8660:	8662:	8663:	8664:	8665:	8667:	8668:
U 08F8	8669:	8671:	8672:	8673:	8675:	8676:	8677:	8679:
U 0900	8680:	8681:	8683:	8684:	8685:	8687:	8688:	8689:
U 0908	8691:	8692:	8693:	8695:	8701:	8702:	8703:	8705:
U 0910	8706:	8708:	8709:	8710:	8712:	8713:	8714:	8716:
U 0918	8955:	8957:	8960:	8962:	8965:	8967:	8968:	8970:
U 0920	8971:	8972:	8973:	8974:	8976:	8978:	8979:	8980:
U 0928	8983:	8984:	8985:	8986:	8987:	8990:	8991:	8992:
U 0930	8993:	8995:	8996:	8998:	8999:	9001:	9002:	9003:
U 0938	9005:	9007:	9009:	9011:	9012:	9013:	9014:	9015:
U 0940	9016:	9017:	9019:	9022:	9023:	9024:	9025:	9026:
U 0948	9029:	9030:	9031:	9032:	9034:	9035:	9037:	9039:
U 0950	9041:	9042:	9043:	9045:	9047:	9049:	9050:	9053:
U 0958	9054:	9056:	9057:	9059:	9061:	9063:	9064:	9067:
U 0960	9068:	9069:	9071:	9072:	9073:	9075:	9076:	9077:
U 0968	9078:	9080:	9082:	9083:	9084:	9085:	9086:	9089:
U 0970	9090:	9091:	9092:	9094:	9095:	9097:	9099:	9101:
U 0978	9102:	9103:	9105:	9107:	9109:	9110:	9113:	9114:
U 0980	9116:	9117:	9119:	9121:	9123:	9124:	9127:	9128:
U 0988	9129:	9130:	9131:	9132:	9133:	9134:	9136:	9139:
U 0990	9142:	9143:	9145:	9147:	9149:	9150:	9151:	9153:
U 0998	9155:	9157:	9158:	9161:	9163:	9166:	9168:	9170:
U 09A0	9171:	9172:	9176:	9177:	9178:	9181:	9182:	9183:
U 09A8	9186:	9187:	9188:	9191:	9192:	9193:	9196:	9197:
U 09B0	9198:	9202:	9203:	9205:	9208:	9209:	9210:	9213:
U 09B8	9214:	9215:	9216:	9218:	9219:	9471:	9473:	9476:
U 09C0	9478:	9481:	9484:	9485:	9486:	9487:	9489:	9492:
U 09C8	9494:	9496:	9498:	9499:	9500:	9501:	9503:	9506:
U 09D0	9507:	9508:	9509:	9510:	9513:	9514:	9515:	9516:
U 09D8	9519:	9520:	9521:	9522:	9524:	9527:	9528:	9531:
U 09E0	9532:	9533:	9534:	9535:	9536:	9538:	9539:	9540:
U 09E8	9543:	9545:	9546:	9549:	9550:	9552:	9553:	9556:
U 09F0	9557:	9558:	9559:	9562:	9563:	9564:	9565:	9567:
U 09F8	9569:	9570:	9571:	9572:	9575:	9576:	9577:	9578:
U 0A00	9581:	9582:	9584:	9585:	9588:	9589:	9590:	9591:
U 0A08	9593:	9594:	9595:	9597:	9598:	9600:	9601:	9604:
U 0A10	9606:	9608:	9609:	9612:	9613:	9614:	9616:	9618:
U 0A18	9619:	9620:	9621:	9624:	9625:	9628:	9629:	9631:
U 0A20	9632:	9634:	9635:	9637:	9638:	9641:	9642:	9643:
U 0A28	9644:	9645:	9648:	9649:	9650:	9651:	9654:	9656:
U 0A30	9657:	9658:	9661:	9662:	9663:	9664:	9666:	9667:
U 0A38	9668:	9671:	9672:	9674:	9675:	9678:	9679:	9680:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 0A40	9681:	9684:	9685:	9686:	9687:	9690:	9692:	9693:
U 0A48	9694:	9697:	9698:	9699:	9700:	9702:	9703:	9704:
U 0A50	9707:	9709:	9710:	9711:	9712:	9713:	9715:	9716:
U 0A58	9718:	9719:	9720:	9723:	9725:	9727:	9729:	9730:
U 0A60	9731:	9734:	9736:	9737:	9740:	9741:	9743:	9744:
U 0A68	9745:	9748:	9750:	9752:	9754:	9755:	9756:	9759:
U 0A70	9760:	9762:	9763:	9765:	9769:	9770:	9771:	9774:
U 0A78	9775:	9776:	9779:	9780:	9781:	9784:	9785:	9786:
U 0A80	9789:	9790:	9791:	9794:	9797:	9798:	9799:	9802:
U 0A88	9803:	9804:	9805:	9807:	9808:	9899:	9901:	9904:
U 0A90	9906:	9909:	9911:	9912:	9915:	9916:	9918:	9919:
U 0A98	9920:	9922:	9923:	9924:	9926:	9927:	10008:	10010:
U 0AA0	10013:	10015:	10018:	10020:	10021:	10023:	10024:	10026:
U 0AA8	10027:	10028:	10030:	10031:	10032:	10034:	10035:	10251:
U 0AB0	10253:	10256:	10258:	10261:	10263:	10264:	10265:	10267:
U 0AB8	10268:	10270:	10271:	10275:	10276:	10278:	10279:	10281:
U 0AC0	10282:	10283:	10285:	10287:	10288:	10290:	10291:	10293:
U 0AC8	10294:	10295:	10297:	10298:	10300:	10303:	10304:	10306:
U 0AD0	10307:	10309:	10310:	10311:	10313:	10314:	10315:	10317:
U 0AD8	10318:	10320:	10321:	10322:	10325:	10326:	10328:	10329:
U 0AE0	10331:	10332:	10334:	10335:	10336:	10338:	10339:	10340:
U 0AE8	10342:	10343:	10345:	10346:	10347:	10348:	10351:	10352:
U 0AF0	10354:	10355:	10357:	10358:	10359:	10361:	10362:	10364:
U 0AF8	10365:	10366:	10369:	10370:	10372:	10373:	10375:	10376:
U 0B00	10377:	10379:	10380:	10382:	10383:	10384:	10386:	10387:
U 0B08	10390:	10391:	10393:	10394:	10396:	10397:	10398:	10400:
U 0B10	10401:	10402:	10404:	10405:	10407:	10408:	10409:	10411:
U 0B18	10412:	10413:	10414:	10416:	10419:	10420:	10423:	10424:
U 0B20	10426:	10427:	10429:	10430:	10431:	10433:	10434:	10435:
U 0B28	10437:	10438:	10440:	10441:	10442:	10444:	10445:	10447:
U 0B30	10449:	10451:	10452:	10454:	10455:	10456:	10458:	10462:
U 0B38	10463:	10465:	10466:	10467:	10470:	10471:	10473:	10474:
U 0B40	10476:	10477:	10478:	10480:	10483:	10484:	10488:	10489:
U 0B48	10490:	10492:	10493:	10494:	10495:	10498:	10499:	10500:
U 0B50	10501:	10502:	10504:	10506:	10508:	10510:	10512:	10513:
U 0B58	10516:	10517:	10518:	10520:	10521:	10523:	10524:	10525:
U 0B60	10527:	10529:	10532:	10533:	10535:	10537:	10538:	10540:
U 0B68	10541:	10544:	10545:	10548:	10549:	10550:	10554:	10556:
U 0B70	10558:	10561:	10562:	10564:	10565:	10566:	10568:	10570:
U 0B78	10572:	10574:	10575:	10577:	10579:	10580:	10582:	10583:
U 0B80	10586:	10587:	10590:	10591:	10592:	10595:	10597:	10598:
U 0B88	10602:	10604:	10605:	10606:	10608:	10609:	10611:	10612:
U 0B90	10613:	10615:	10617:	10620:	10621:	10625:	10626:	10628:
U 0B98	10630:	10632:	10634:	10635:	10637:	10638:	10639:	10641:
U 0BA0	10643:	10645:	10647:	10648:	10652:	10653:	10810:	10812:
U 0BA8	10815:	10817:	10820:	10822:	10823:	10824:	10825:	10826:
U 0BB0	10828:	10829:	10830:	10832:	10833:	10837:	10838:	10840:
U 0BB8	10841:	10843:	10844:	10845:	10847:	10849:	10850:	10852:
U 0BC0	10853:	10855:	10856:	10857:	10859:	10860:	10862:	10865:
U 0BC8	10866:	10868:	10869:	10871:	10872:	10873:	10875:	10876:
U 0BD0	10877:	10879:	10880:	10882:	10883:	10884:	10887:	10888:
U 0BD8	10890:	10891:	10893:	10894:	10896:	10897:	10898:	10900:
U 0BE0	10901:	10902:	10904:	10905:	10907:	10908:	10911:	10912:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 0BE8	10914:	10915:	10917:	10918:	10919:	10921:	10922:	10924:
U 0BF0	10925:	10927:	10928:	10931:	10932:	10934:	10935:	10937:
U 0BF8	10938:	10939:	10941:	10942:	10944:	10945:	10946:	10948:
U 0C00	10949:	10950:	10953:	10954:	10956:	10957:	10959:	10960:
U 0C08	10961:	10963:	10964:	10965:	10967:	10968:	10970:	10971:
U 0C10	10972:	10974:	10975:	10976:	10977:	10978:	10980:	10983:
U 0C18	10984:	10987:	10988:	10990:	10991:	10993:	10994:	10995:
U 0C20	10997:	10998:	10999:	11001:	11002:	11004:	11005:	11006:
U 0C28	11008:	11009:	11011:	11013:	11015:	11016:	11018:	11019:
U 0C30	11021:	11022:	11024:	11029:	11030:	11032:	11033:	11034:
U 0C38	11037:	11038:	11040:	11041:	11043:	11044:	11045:	11047:
U 0C40	11048:	11049:	11050:	11051:	11052:	11056:	11057:	11061:
U 0C48	11062:	11063:	11064:	11065:	11067:	11069:	11071:	11073:
U 0C50	11075:	11076:	11079:	11080:	11081:	11083:	11084:	11086:
U 0C58	11087:	11088:	11090:	11092:	11094:	11095:	11097:	11098:
U 0C60	11100:	11102:	11103:	11105:	11106:	11110:	11112:	11114:
U 0C68	11117:	11118:	11120:	11121:	11122:	11124:	11126:	11128:
U 0C70	11129:	11130:	11132:	11133:	11135:	11137:	11138:	11140:
U 0C78	11141:	11144:	11146:	11147:	11219:	11221:	11224:	11226:
U 0C80	11229:	11231:	11232:	11233:	11234:	11236:	11237:	11238:
U 0C88	11240:	11242:	11243:	11244:	11246:	11249:	11250:	11251:
U 0C90	11253:	11254:	11255:	11257:	11258:	11259:	11261:	11262:
U 0C98	11263:	11265:	11266:	11267:	11270:	11271:	11272:	11273:
U 0CA0	11275:	11277:	11279:	11280:	11281:	11283:	11284:	11285:
U 0CA8	11286:	11288:	11289:	11290:	11360:	11362:	11365:	11367:
U 0CB0	11370:	11372:	11373:	11374:	11375:	11376:	11377:	11378:
U 0CB8	11379:	11382:	11384:	11386:	11387:	11388:	11390:	11393:
U 0CC0	11395:	11396:	11397:	11399:	11400:	11402:	11403:	11405:
U 0CC8	11406:	11407:	11409:	11410:	11412:	11413:	11415:	11416:
U 0CD0	11417:	11419:	11420:	11424:	11425:	11427:	11428:	11431:
U 0CD8	11432:	11434:	11435:	11436:	11438:	11439:	11441:	11442:
U 0CE0	11444:	11445:	11446:	11448:	11449:	11452:	11453:	11455:
U 0CE8	11456:	11457:	11460:	11462:	11463:	11465:	11466:	11467:
U 0CF0	11469:	11470:	11472:	11473:	11475:	11476:	11477:	11479:
U 0CF8	11480:	11481:	11482:	11483:	11486:	11487:	11488:	11489:
U 0D00	11490:	11492:	11494:	11495:	11497:	11498:	11499:	11501:
U 0D08	11502:	11504:	11505:	11507:	11508:	11509:	11511:	11512:
U 0D10	11513:	11514:	11515:	11734:	11736:	11739:	11741:	11744:
U 0D18	11746:	11747:	11749:	11750:	11752:	11753:	11754:	11755:
U 0D20	11756:	11759:	11763:	11764:	11765:	11768:	11770:	11771:
U 0D28	11772:	11774:	11775:	11776:	11778:	11780:	11781:	11782:
U 0D30	11783:	11785:	11786:	11788:	11790:	11793:	11794:	11797:
U 0D38	11799:	11801:	11802:	11803:	11805:	11807:	11808:	11811:
U 0D40	11812:	11813:	11814:	11815:	11816:	11817:	11820:	11822:
U 0D48	11823:	11824:	11825:	11827:	11830:	11832:	11834:	11835:
U 0D50	11837:	11838:	11839:	11840:	11841:	11843:	11845:	11846:
U 0D58	11847:	11848:	11850:	11851:	11853:	11855:	11857:	11858:
U 0D60	11859:	11860:	11863:	11864:	11867:	11869:	11871:	11872:
U 0D68	11873:	11875:	11877:	11878:	11881:	11882:	11883:	11885:
U 0D70	11887:	11888:	11889:	11891:	11894:	11896:	11897:	11900:
U 0D78	11902:	11904:	11905:	11906:	11908:	11910:	11911:	11913:
U 0D80	11914:	11918:	11919:	11920:	11922:	11924:	11925:	11927:
U 0D88	11928:	11929:	11930:	11932:	11935:	11937:	11938:	11941:

;
; Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 0D90	11943:	11945:	11946:	11947:	11949:	11951:	11952:	11954:
U 0D98	11955:	11959:	11960:	11962:	11964:	11966:	11967:	11970:
U 0DA0	11971:	11972:	11975:	11979:	11980:	11981:	11983:	11986:
U 0DA8	11988:	11990:	11992:	11994:	11995:	11996:	11999:	12001:
U 0DB0	12003:	12005:	12007:	12009:	12011:	12012:	12013:	12015:
U 0DB8	12018:	12019:	12023:	12024:	12025:	12027:	12028:	12029:
U 0DC0	12030:	12033:	12034:	12035:	12037:	12039:	12041:	12043:
U 0DC8	12044:	12048:	12049:	12051:	12054:	12056:	12057:	12058:
U 0DD0	12059:	12061:	12062:	12064:	12066:	12067:	12068:	12069:
U 0DD8	12071:	12073:	12076:	12077:	12079:	12081:	12082:	12084:
U 0DE0	12085:	12088:	12089:	12092:	12093:	12094:	12098:	12101:
U 0DE8	12103:	12104:	12105:	12106:	12109:	12110:	12112:	12114:
U 0DF0	12116:	12117:	12118:	12120:	12122:	12124:	12126:	12127:
U 0DF8	12129:	12131:	12132:	12134:	12135:	12138:	12139:	12142:
U 0E00	12143:	12144:	12147:	12149:	12150:	12154:	12157:	12159:
U 0E08	12160:	12161:	12162:	12164:	12165:	12167:	12169:	12170:
U 0E10	12171:	12172:	12174:	12176:	12179:	12180:	12184:	12185:
U 0E18	12188:	12191:	12193:	12194:	12195:	12196:	12199:	12200:
U 0E20	12202:	12204:	12206:	12207:	12208:	12210:	12212:	12214:
U 0E28	12216:	12217:	12221:	12222:	12225:	12226:	12227:	12228:
U 0E30	12229:	12232:	12234:	12238:	12239:	12240:	12242:	12244:
U 0E38	12246:	12247:	12248:	12250:	12251:	12254:	12258:	12260:
U 0E40	12262:	12264:	12266:	12267:	12268:	12464:	12466:	12469:
U 0E48	12471:	12474:	12476:	12477:	12478:	12479:	12480:	12482:
U 0E50	12483:	12485:	12486:	12488:	12489:	12490:	12492:	12493:
U 0E58	12496:	12497:	12498:	12499:	12501:	12502:	12505:	12507:
U 0E60	12510:	12511:	12515:	12516:	12517:	12518:	12521:	12522:
U 0E68	12524:	12525:	12528:	12530:	12531:	12533:	12536:	12537:
U 0E70	12541:	12542:	12543:	12544:	12547:	12548:	12550:	12551:
U 0E78	12554:	12556:	12559:	12560:	12563:	12564:	12566:	12567:
U 0E80	12570:	12572:	12573:	12575:	12578:	12579:	12582:	12583:
U 0E88	12584:	12585:	12586:	12587:	12589:	12590:	12592:	12593:
U 0E90	12594:	12596:	12597:	12598:	12600:	12601:	12602:	12603:
U 0E98	12604:	12607:	12608:	12609:	12611:	12612:	12614:	12616:
U 0EA0	12617:	12618:	12620:	12621:	12622:	12623:	12625:	12627:
U 0EA8	12629:	12630:	12632:	12636:	12637:	12639:	12641:	12642:
U 0EB0	12644:	12646:	12648:	12649:	12650:	12651:	12655:	12656:
U 0EB8	12657:	12659:	12660:	12662:	12664:	12666:	12667:	12668:
U 0EC0	12669:	12738:	12740:	12743:	12745:	12748:	12750:	12751:
U 0EC8	12752:	12754:	12756:	12758:	12759:	12761:	12762:	12764:
U 0ED0	12765:	12766:	12768:	12769:	12771:	12773:	12774:	12776:
U 0ED8	12777:	12779:	12780:	12781:	12859:	12861:	12864:	12866:
U 0EE0	12869:	12871:	12872:	12873:	12875:	12877:	12879:	12880:
U 0EE8	12882:	12884:	12886:	12887:	12888:	12890:	12891:	12892:
U 0EF0	12893:	12895:	12896:	12898:	12900:	12902:	12903:	12904:
U 0EF8	13101:	13103:	13106:	13108:	13111:	13113:	13114:	13115:
U 0F00	13117:	13118:	13120:	13121:	13122:	13124:	13125:	13126:
U 0F08	13128:	13130:	13133:	13135:	13136:	13138:	13139:	13141:
U 0F10	13142:	13144:	13146:	13147:	13150:	13151:	13152:	13154:
U 0F18	13155:	13158:	13159:	13161:	13162:	13164:	13165:	13166:
U 0F20	13167:	13168:	13169:	13170:	13173:	13174:	13177:	13180:
U 0F28	13181:	13183:	13185:	13187:	13189:	13191:	13192:	13194:
U 0F30	13196:	13197:	13200:	13201:	13202:	13204:	13205:	13208:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 0F38	13209:	13211:	13212:	13213:	13214:	13215:	13216:	13217:
U 0F40	13220:	13221:	13224:	13227:	13228:	13230:	13232:	13234:
U 0F48	13236:	13238:	13239:	13241:	13243:	13244:	13247:	13248:
U 0F50	13249:	13251:	13252:	13255:	13259:	13260:	13261:	13263:
U 0F58	13264:	13265:	13267:	13268:	13269:	13271:	13276:	13278:
U 0F60	13279:	13281:	13283:	13284:	13286:	13287:	13288:	13290:
U 0F68	13294:	13295:	13298:	13299:	13301:	13303:	13304:	13305:
U 0F70	13307:	13309:	13441:	13443:	13446:	13448:	13451:	13453:
U 0F78	13454:	13455:	13457:	13458:	13459:	13461:	13462:	13464:
U 0F80	13465:	13467:	13468:	13469:	13471:	13472:	13473:	13475:
U 0F88	13476:	13478:	13479:	13481:	13482:	13484:	13485:	13487:
U 0F90	13489:	13490:	13491:	13492:	13493:	13495:	13496:	13498:
U 0F98	13500:	13501:	13503:	13504:	13506:	13507:	13509:	13510:
U 0FA0	13512:	13514:	13515:	13516:	13517:	13519:	13520:	13522:
U 0FA8	13523:	13524:	13525:	13527:	13528:	13530:	13531:	13533:
U 0FB0	13534:	13536:	13715:	13717:	13720:	13722:	13725:	13727:
U 0FB8	13728:	13730:	13731:	13732:	13733:	13734:	13735:	13737:
U 0FC0	13738:	13740:	13741:	13743:	13744:	13745:	13747:	13749:
U 0FC8	13750:	13752:	13753:	13755:	13756:	13757:	13759:	13761:
U 0FD0	13762:	13764:	13766:	13768:	13769:	13770:	13772:	13774:
U 0FD8	13775:	13777:	13779:	13781:	13782:	13783:	13785:	13788:
U 0FE0	13789:	13791:	13793:	13795:	13796:	13797:	13799:	13801:
U 0FE8	13802:	13803:	13804:	13807:	13809:	13810:	13811:	13813:
U 0FF0	13814:	13815:	13817:	13819:	13820:	13823:	13825:	13827:
U 0FF8	13828:	13829:	13832:					
U 1000	14011:	14013:	14016:	14018:	14021:	14023:	14024:	14027:
U 1008	14028:	14030:	14031:	14032:	14033:	14034:	14035:	14036:
U 1010	14037:	14039:	14040:	14041:	14042:	14043:	14045:	14046:
U 1018	14047:	14049:	14050:	14051:	14053:	14054:	14055:	14057:
U 1020	14058:	14061:	14063:	14065:	14066:	14067:	14069:	14070:
U 1028	14071:	14073:	14074:	14076:	14078:	14080:	14081:	14082:
U 1030	14084:	14085:	14086:	14088:	14090:	14091:	14092:	14093:
U 1038	14095:	14096:	14099:	14101:	14103:	14104:	14105:	14108:
U 1040	14109:	14111:	14112:	14113:	14114:	14115:	14116:	14117:
U 1048	14118:	14120:	14121:	14124:	14125:	14127:	14128:	14129:
U 1050	14131:	14132:	14133:	14134:	14135:	14136:	14138:	14139:
U 1058	14142:	14143:	14145:	14146:	14147:	14148:	14152:	14153:
U 1060	14156:	14158:	14160:	14161:	14162:	14163:	14284:	14286:
U 1068	14289:	14291:	14294:	14297:	14298:	14299:	14301:	14302:
U 1070	14304:	14305:	14306:	14307:	14309:	14311:	14312:	14313:
U 1078	14314:	14315:	14318:	14319:	14320:	14321:	14323:	14324:
U 1080	14325:	14327:	14328:	14329:	14330:	14331:	14332:	14333:
U 1088	14334:	14336:	14337:	14339:	14341:	14342:	14344:	14346:
U 1090	14349:	14350:	14352:	14353:	14355:	14357:	14358:	14360:
U 1098	14362:	14363:	14364:	14366:	14369:	14370:	14372:	14373:
U 10A0	14375:	14376:	14378:	14379:	14381:	14384:	14385:	14386:
U 10A8	14387:	14388:	14389:	14390:	14392:	14393:	14395:	14396:
U 10B0	14398:	14399:	14400:	14402:	14403:	14404:	14405:	14406:
U 10B8	14408:	14409:	14519:	14521:	14524:	14526:	14529:	14531:
U 10C0	14532:	14536:	14537:	14539:	14543:	14544:	14545:	14546:
U 10C8	14547:	14549:	14550:	14552:	14554:	14555:	14557:	14558:
U 10D0	14559:	14561:	14562:	14564:	14565:	14567:	14568:	14571:
U 10D8	14573:	14574:	14576:	14577:	14578:	14579:	14582:	14584:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 10E0	14586:	14587:	14589:	14590:	14592:	14594:	14595:	14596:
U 10E8	14598:	14599:	14601:	14602:	14604:	14605:	14606:	14609:
U 10F0	14610:	14611:	14612:	14613:	14615:	14616:	14617:	14618:
U 10F8	14619:	14620:	14623:	14624:	14627:	14629:	14630:	14632:
U 1100	14633:	14634:	14635:	14636:	14639:	14641:	14643:	14644:
U 1108	14646:	14647:	14649:	14650:	14651:	14653:	14654:	14656:
U 1110	14657:	14659:	14660:	14661:	14664:	14665:	14666:	14667:
U 1118	14668:	14670:	14671:	14672:	14673:	14674:	14675:	14731:
U 1120	14733:	14736:	14738:	14741:	14743:	14744:	14748:	14749:
U 1128	14751:	14755:	14756:	14758:	14759:	14761:	14763:	14764:
U 1130	14766:	14767:	14769:	14770:	14771:	14773:	14774:	14776:
U 1138	14777:	14779:	14780:	14781:	14782:	14785:	14786:	14858:
U 1140	14860:	14863:	14865:	14868:	14870:	14871:	14872:	14876:
U 1148	14877:	14879:	14883:	14884:	14886:	14888:	14889:	14891:
U 1150	14892:	14894:	14895:	14897:	14899:	14901:	14902:	14904:
U 1158	14905:	14907:	14908:	14910:	14911:	14912:	14914:	14915:
U 1160	14917:	14918:	14920:	14922:	14925:	14927:	14929:	14931:
U 1168	14932:	14934:	14935:	14937:	14938:	14940:	14942:	14944:
U 1170	14945:	14947:	14948:	14950:	14951:	14953:	14954:	14955:
U 1178	14957:	14958:	14960:	14961:	14962:	14969:	14970:	14971:
U 1180	14975:	14976:	14977:	14978:	14981:	14983:	15159:	15161:
U 1188	15164:	15166:	15169:	15171:	15172:	15176:	15177:	15179:
U 1190	15182:	15183:	15185:	15187:	15188:	15190:	15191:	15193:
U 1198	15194:	15195:	15197:	15198:	15199:	15202:	15203:	15204:
U 11A0	15206:	15207:	15209:	15210:	15211:	15213:	15214:	15215:
U 11A8	15217:	15220:	15221:	15222:	15223:	15225:	15226:	15227:
U 11B0	15229:	15231:	15232:	15233:	15235:	15236:	15238:	15239:
U 11B8	15240:	15242:	15243:	15245:	15246:	15247:	15249:	15252:
U 11C0	15253:	15254:	15255:	15258:	15259:	15260:	15262:	15265:
U 11C8	15266:	15268:	15269:	15272:	15273:	15274:	15275:	15276:
U 11D0	15278:	15279:	15281:	15282:	15284:	15285:	15286:	15289:
U 11D8	15290:	15292:	15293:	15295:	15296:	15297:	15300:	15301:
U 11E0	15304:	15305:	15306:	15309:	15310:	15311:	15313:	15314:
U 11E8	15316:	15317:	15319:	15320:	15321:	15323:	15326:	15327:
U 11F0	15328:	15329:	15332:	15333:	15334:	15336:	15338:	15339:
U 11F8	15340:	15342:	15343:	15345:	15346:	15349:	15350:	15352:
U 1200	15353:	15354:	15356:	15359:	15360:	15361:	15362:	15365:
U 1208	15366:	15367:	15369:	15372:	15373:	15375:	15376:	15379:
U 1210	15380:	15383:	15384:	15386:	15388:	15391:	15393:	15394:
U 1218	15396:	15398:	15399:	15401:	15403:	15404:	15406:	15407:
U 1220	15408:	15412:	15414:	15415:	15416:	15423:	15424:	15425:
U 1228	15426:	15427:	15428:	15430:	15431:	15432:	15434:	15435:
U 1230	15436:	15437:	15439:	15442:	15443:	15445:	15447:	15449:
U 1238	15452:	15456:	15457:	15458:	15459:	15460:	15461:	15463:
U 1240	15464:	15465:	15467:	15468:	15469:	15470:	15472:	15475:
U 1248	15476:	15478:	15480:	15482:	15485:	15489:	15491:	15492:
U 1250	15493:	15495:	15496:	15497:	15500:	15501:	15503:	15504:
U 1258	15505:	15507:	15508:	15510:	15511:	15513:	15516:	15517:
U 1260	15518:	15520:	15521:	15522:	15523:	15526:	15527:	15529:
U 1268	15530:	15533:	15534:	15535:	15537:	15538:	15541:	15542:
U 1270	15544:	15546:	15547:	15549:	15551:	15552:	15553:	15554:
U 1278	15555:	15626:	15628:	15631:	15633:	15636:	15639:	15640:
U 1280	15641:	15642:	15643:	15646:	15647:	15649:	15651:	15653:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 1288	15654:	15655:	15657:	15659:	15660:	15662:	15664:	15666:
U 1290	15667:	15669:	15754:	15756:	15759:	15761:	15764:	15766:
U 1298	15767:	15770:	15771:	15772:	15773:	15775:	15776:	15777:
U 12A0	15778:	15780:	15782:	15783:	15786:	15788:	15792:	15793:
U 12A8	15795:	15796:	15798:	15799:	15800:	15802:	15804:	15805:
U 12B0 - 12BF Unused								
U 12C0	15808:	15810:	15811:	15813:	15814:	15815:	15817:	15818:
U 12C8	15819:	15820:	15821:	15823:	15824:	15827:	15829:	15972:
U 12D0	15974:	15977:	15979:	15982:	15984:	15985:	15987:	15988:
U 12D8	15989:	15990:	15991:	15992:	15993:	15995:	15996:	15997:
U 12E0	15998:	16000:	16001:	16002:	16003:	16005:	16006:	16007:
U 12E8	16008:	16009:	16011:	16013:	16015:	16016:	16018:	16019:
U 12F0	16020:	16022:	16024:	16025:	16027:	16028:	16030:	16031:
U 12F8	16033:	16034:	16035:	16037:	16038:	16039:	16040:	16041:
U 1300	16044:	16047:	16048:	16049:	16050:	16052:	16053:	16055:
U 1308	16056:	16058:	16059:	16061:	16062:	16063:	16065:	16066:
U 1310	16067:	16068:	16069:	16072:	16075:	16076:	16077:	16079:
U 1318	16080:	16082:	16083:	16085:	16086:	16088:	16089:	16090:
U 1320	16092:	16093:	16094:	16095:	16096:	16099:	16102:	16103:
U 1328	16104:	16105:	16106:	16107:	16108:	16109:	16111:	16113:
U 1330	16114:	16115:	16118:	16119:	16121:	16122:	16124:	16125:
U 1338	16127:	16128:	16129:	16131:	16132:	16133:	16134:	16135:
U 1340	16138:	16141:	16142:	16143:	16144:	16145:	16148:	16149:
U 1348	16151:	16152:	16154:	16155:	16157:	16158:	16159:	16161:
U 1350	16162:	16163:	16164:	16165:	16168:	16171:	16172:	16173:
U 1358	16174:	16175:	16176:	16178:	16181:	16182:	16184:	16185:
U 1360	16187:	16188:	16190:	16191:	16192:	16194:	16195:	16196:
U 1368	16197:	16198:	16201:	16204:	16205:	16206:	16207:	16208:
U 1370	16210:	16212:	16213:	16215:	16216:	16218:	16219:	16221:
U 1378	16222:	16223:	16225:	16226:	16227:	16228:	16229:	16232:
U 1380	16235:	16236:	16237:	16238:	16239:	16241:	16243:	16244:
U 1388	16246:	16247:	16249:	16250:	16252:	16253:	16254:	16256:
U 1390	16257:	16258:	16259:	16260:	16263:	16266:	16267:	16268:
U 1398	16269:	16270:	16272:	16274:	16275:	16277:	16278:	16280:
U 13A0	16281:	16283:	16284:	16285:	16287:	16288:	16289:	16290:
U 13A8	16291:	16294:	16296:	16300:	16301:	16303:	16304:	16305:
U 13B0	16306:	16309:	16310:	16312:	16313:	16314:	16317:	16318:
U 13B8	16320:	16321:	16322:	16323:	16326:	16327:	16329:	16330:
U 13C0	16332:	16333:	16335:	16336:	16337:	16338:	16462:	16464:
U 13C8	16467:	16469:	16472:	16475:	16476:	16478:	16479:	16480:
U 13D0	16482:	16483:	16485:	16487:	16488:	16490:	16491:	16493:
U 13D8	16494:	16495:	16496:	16497:	16499:	16500:	16502:	16503:
U 13E0	16504:	16505:	16507:	16508:	16510:	16512:	16513:	16515:
U 13E8	16516:	16518:	16519:	16520:	16522:	16523:	16525:	16526:
U 13F0	16529:	16531:	16532:	16534:	16536:	16537:	16539:	16540:
U 13F8	16542:	16543:	16545:	16546:	16547:	16549:	16550:	16551:
U 1400	16553:	16554:	16556:	16557:	16558:	16560:	16561:	16562:
U 1408	16564:	16566:	16567:	16569:	16570:	16571:	16572:	16574:
U 1410	16575:	16576:	16578:	16579:	16580:	16581:	16582:	16585:
U 1418	16586:	16588:	16590:	16591:	16593:	16595:	16596:	16598:
U 1420	16599:	16601:	16603:	16605:	16606:	16607:	16609:	16610:
U 1428	16611:	16613:	16614:	16616:	16617:	16618:	16620:	16621:
U 1430	16624:	16625:	16627:	16628:	16630:	16631:	16632:	16634:

;
; Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 1438	16636:	16637:	16639:	16640:	16641:	16643:	16644:	16645:
U 1440	16646:	16650:	16651:	16652:	16654:	16656:	16657:	16658:
U 1448	16745:	16747:	16750:	16752:	16755:	16757:	16759:	16760:
U 1450	16761:	16762:	16763:	16765:	16766:	16768:	16769:	16771:
U 1458	16773:	16774:	16776:	16777:	16778:	16780:	16781:	16783:
U 1460	16784:	16786:	16788:	16789:	16791:	16792:	16793:	16794:
U 1468	16796:	16799:	16800:	16802:	16803:	16805:	16806:	16807:
U 1470	16809:	16811:	16812:	16814:	16815:	16817:	16818:	16819:
U 1478	16958:	16960:	16963:	16965:	16968:	16970:	16971:	16973:
U 1480	16974:	16976:	16977:	16979:	16980:	16982:	16984:	16986:
U 1488	16987:	16988:	16990:	16991:	16992:	16993:	16994:	16995:
U 1490	16996:	16998:	17000:	17001:	17003:	17004:	17005:	17007:
U 1498	17008:	17010:	17012:	17013:	17014:	17015:	17017:	17019:
U 14A0	17020:	17021:	17022:	17024:	17025:	17027:	17029:	17030:
U 14A8	17031:	17032:	17034:	17035:	17036:	17038:	17039:	17042:
U 14B0	17044:	17046:	17047:	17049:	17050:	17052:	17053:	17054:
U 14B8	17055:	17057:	17058:	17059:	17060:	17062:	17063:	17064:
U 14C0	17065:	17066:	17067:	17069:	17070:	17072:	17074:	17075:
U 14C8	17076:	17077:	17079:	17080:	17081:	17082:	17084:	17087:
U 14D0	17088:	17089:	17090:	17092:	17094:	17095:	17096:	17097:
U 14D8	17098:	17099:	17191:	17193:	17196:	17198:	17201:	17203:
U 14E0	17204:	17205:	17208:	17211:	17212:	17213:	17214:	17215:
U 14E8	17217:	17218:	17219:	17220:	17222:	17225:	17226:	17227:
U 14F0	17228:	17231:	17232:	17233:	17234:	17235:	17238:	17239:
U 14F8	17240:	17243:	17244:	17245:	17246:	17247:	17251:	17252:
U 1500	17254:	17255:	17256:	17258:	17259:	17260:	17262:	17264:
U 1508	17265:	17266:	17268:	17269:	17270:	17271:	17272:	17273:
U 1510	17275:	17277:	17278:	17279:	17372:	17374:	17377:	17379:
U 1518	17382:	17384:	17385:	17388:	17389:	17391:	17393:	17394:
U 1520	17395:	17399:	17400:	17401:	17402:	17404:	17406:	17409:
U 1528	17411:	17412:	17413:	17414:	17415:	17417:	17419:	17421:
U 1530	17422:	17425:	17427:	17428:	17429:	17431:	17432:	17434:
U 1538	17436:	17439:	17440:	17441:	17442:	17443:	17444:	17446:
U 1540	17447:	17449:	17450:	17451:	17453:	17454:	17456:	17457:
U 1548	17458:	17462:	17463:	17464:	17465:	17467:	17469:	17472:
U 1550	17474:	17475:	17476:	17477:	17478:	17480:	17482:	17484:
U 1558	17485:	17488:	17489:	17490:	17491:	17493:	17494:	17496:
U 1560	17498:	17501:	17502:	17503:	17504:	17505:	17506:	17508:
U 1568	17509:	17511:	17512:	17513:	17574:	17576:	17579:	17581:
U 1570	17584:	17587:	17588:	17589:	17591:	17592:	17594:	17596:
U 1578	17598:	17600:	17601:	17602:	17603:	17605:	17607:	17609:
U 1580	17611:	17612:	17613:	17614:	17616:	17618:	17621:	17622:
U 1588	17623:	17624:	17740:	17742:	17745:	17747:	17750:	17752:
U 1590	17753:	17755:	17756:	17757:	17758:	17759:	17761:	17763:
U 1598	17764:	17765:	17766:	17767:	17769:	17772:	17775:	17776:
U 15A0	17777:	17779:	17780:	17782:	17783:	17784:	17786:	17787:
U 15A8	17790:	17791:	17792:	17794:	17795:	17797:	17799:	17800:
U 15B0	17801:	17802:	17804:	17805:	17808:	17809:	17810:	17811:
U 15B8	17813:	17814:	17817:	17818:	17819:	17821:	17822:	17824:
U 15C0	17825:	17826:	17827:	17830:	17833:	17836:	17837:	17838:
U 15C8	17839:	17841:	17842:	17844:	17845:	17846:	17848:	17849:
U 15D0	17852:	17853:	17854:	17855:	17856:	17858:	17859:	17861:
U 15D8	17863:	17864:	17865:	17866:	17868:	17869:	17872:	17873:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 15E0	17874:	17876:	17877:	17880:	17881:	17882:	17884:	17885:
U 15E8	17887:	17891:	17892:	17893:	17894:	17896:	17897:	17898:
U 15F0	17900:	17901:	17902:	17903:	17905:	17906:	17907:	17908:
U 15F8	17912:	17913:	17914:	17915:	17917:	17918:	17919:	17921:
U 1600	17922:	17923:	17924:	17925:	17927:	17928:	17929:	17930:
U 1608	18120:	18122:	18125:	18127:	18130:	18132:	18133:	18134:
U 1610	18135:	18139:	18141:	18142:	18144:	18145:	18146:	18147:
U 1618	18149:	18150:	18151:	18152:	18153:	18154:	18156:	18158:
U 1620	18159:	18160:	18161:	18163:	18164:	18165:	18166:	18167:
U 1628	18169:	18170:	18171:	18173:	18175:	18176:	18177:	18178:
U 1630	18180:	18181:	18182:	18183:	18185:	18186:	18187:	18189:
U 1638	18190:	18192:	18194:	18195:	18197:	18198:	18199:	18201:
U 1640	18203:	18204:	18205:	18206:	18208:	18209:	18210:	18211:
U 1648	18213:	18214:	18215:	18217:	18218:	18219:	18221:	18222:
U 1650	18223:	18224:	18225:	18227:	18228:	18229:	18231:	18233:
U 1658	18234:	18235:	18236:	18238:	18239:	18240:	18242:	18243:
U 1660	18244:	18245:	18246:	18248:	18249:	18250:	18252:	18253:
U 1668	18254:	18255:	18257:	18258:	18259:	18260:	18261:	18263:
U 1670	18266:	18268:	18269:	18270:	18271:	18273:	18274:	18275:
U 1678	18277:	18279:	18280:	18281:	18282:	18284:	18285:	18286:
U 1680	18288:	18289:	18290:	18291:	18292:	18294:	18295:	18296:
U 1688	18298:	18299:	18300:	18301:	18303:	18304:	18305:	18307:
U 1690	18309:	18310:	18311:	18312:	18313:	18314:	18315:	18316:
U 1698	18318:	18319:	18320:	18322:	18324:	18325:	18326:	18327:
U 16A0	18329:	18330:	18331:	18333:	18334:	18335:	18336:	18337:
U 16A8	18339:	18340:	18341:	18343:	18344:	18345:	18346:	18348:
U 16B0	18349:	18350:	18351:	18352:	18353:	18354:	18355:	18356:
U 16B8	18358:	18359:	18360:	18362:	18364:	18365:	18367:	18368:
U 16C0	18369:	18370:	18371:	18373:	18374:	18375:	18377:	18378:
U 16C8	18379:	18380:	18484:	18486:	18489:	18491:	18494:	18496:
U 16D0	18497:	18499:	18501:	18503:	18504:	18505:	18507:	18508:
U 16D8	18509:	18511:	18512:	18514:	18516:	18517:	18519:	18520:
U 16E0	18521:	18523:	18524:	18526:	18528:	18529:	18531:	18532:
U 16E8	18533:	18535:	18536:	18538:	18540:	18541:	18543:	18544:
U 16F0	18545:	18547:	18548:	18550:	18552:	18553:	18555:	18556:
U 16F8	18557:	18559:	18560:	18562:	18564:	18565:	18567:	18568:
U 1700	18569:	18571:	18572:	18574:	18576:	18577:	18579:	18580:
U 1708	18581:	18583:	18584:	18586:	18588:	18589:	18591:	18592:
U 1710	18593:	18595:	18596:	18598:	18600:	18601:	18603:	18604:
U 1718	18605:	18695:	18697:	18700:	18702:	18705:	18707:	18708:
U 1720	18710:	18711:	18713:	18715:	18716:	18719:	18720:	18721:
U 1728	18722:	18723:	18725:	18728:	18729:	18730:	18731:	18733:
U 1730	18736:	18738:	18740:	18741:	18742:	18743:	18745:	18746:
U 1738	18747:	18749:	18750:	18751:	18752:	18753:	18755:	18756:
U 1740	18757:	18758:	18760:	18761:	18762:	18764:	18765:	18766:
U 1748	18767:	18768:	18769:	18770:	18772:	18773:	18774:	18776:
U 1750	18778:	18779:	18780:	18781:	18782:	18784:	18785:	18786:
U 1758	18788:	18789:	18790:	18792:	18793:	18794:	18795:	18796:
U 1760	18798:	18799:	18800:	18801:	18803:	18804:	18805:	18887:
U 1768	18889:	18892:	18894:	18897:	18900:	18902:	18903:	18904:
U 1770	18905:	18906:	18908:	18910:	18911:	18912:	18913:	18914:
U 1778	18915:	18916:	18918:	18921:	18923:	18924:	18926:	18927:
U 1780	18929:	18930:	18931:	18933:	18935:	18936:	18938:	18939:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 1788	18941:	18942:	18943:	18945:	18946:	18947:	18948:	18950:
U 1790	18951:	18953:	18954:	18956:	18957:	18958:	19132:	19134:
U 1798	19137:	19139:	19142:	19144:	19146:	19147:	19148:	19151:
U 17A0	19152:	19154:	19155:	19156:	19158:	19159:	19160:	19161:
U 17A8	19162:	19164:	19165:	19166:	19167:	19169:	19170:	19171:
U 17B0	19173:	19174:	19176:	19177:	19179:	19180:	19181:	19183:
U 17B8	19184:	19185:	19186:	19187:	19189:	19190:	19191:	19192:
U 17C0	19194:	19195:	19196:	19198:	19199:	19201:	19202:	19204:
U 17C8	19205:	19206:	19208:	19210:	19211:	19212:	19213:	19214:
U 17D0	19216:	19217:	19218:	19219:	19221:	19222:	19223:	19225:
U 17D8	19226:	19228:	19229:	19231:	19232:	19233:	19235:	19237:
U 17E0	19238:	19239:	19240:	19241:	19243:	19244:	19245:	19246:
U 17E8	19248:	19249:	19250:	19252:	19253:	19254:	19255:	19257:
U 17F0	19260:	19262:	19263:	19264:	19265:	19268:	19271:	19272:
U 17F8	19273:	19275:	19277:	19278:	19280:	19281:	19283:	19284:
U 1800	19285:	19287:	19288:	19289:	19291:	19292:	19293:	19295:
U 1808	19296:	19297:	19298:	19300:	19303:	19305:	19308:	19311:
U 1810	19312:	19313:	19315:	19316:	19318:	19320:	19321:	19323:
U 1818	19324:	19325:	19327:	19328:	19329:	19331:	19332:	19333:
U 1820	19335:	19336:	19337:	19339:	19340:	19341:	19344:	19346:
U 1828	19348:	19350:	19352:	19353:	19355:	19357:	19359:	19360:
U 1830	19361:	19363:	19364:	19365:	19367:	19368:	19369:	19371:
U 1838	19372:	19375:	19378:	19379:	19381:	19382:	19384:	19386:
U 1840	19387:	19388:	19390:	19391:	19392:	19395:	19397:	19399:
U 1848	19401:	19403:	19405:	19407:	19408:	19410:	19411:	19412:
U 1850	19414:	19415:	19417:	19419:	19420:	19421:	19579:	19581:
U 1858	19584:	19586:	19589:	19591:	19592:	19593:	19595:	19597:
U 1860	19599:	19601:	19602:	19603:	19604:	19606:	19607:	19608:
U 1868	19610:	19611:	19613:	19614:	19615:	19617:	19618:	19619:
U 1870	19621:	19622:	19624:	19625:	19627:	19629:	19630:	19631:
U 1878	19633:	19634:	19635:	19636:	19638:	19639:	19641:	19642:
U 1880	19643:	19645:	19646:	19648:	19649:	19650:	19652:	19653:
U 1888	19654:	19656:	19657:	19659:	19660:	19661:	19662:	19663:
U 1890	19664:	19665:	19666:	19668:	19671:	19672:	19673:	19674:
U 1898	19675:	19678:	19680:	19681:	19682:	19683:	19685:	19686:
U 18A0	19687:	19688:	19690:	19691:	19693:	19694:	19695:	19697:
U 18A8	19698:	19699:	19701:	19702:	19704:	19705:	19706:	19707:
U 18B0	19708:	19709:	19710:	19713:	19715:	19716:	19717:	19718:
U 18B8	19720:	19721:	19722:	19724:	19725:	19727:	19728:	19729:
U 18C0	19731:	19732:	19734:	19735:	19737:	19738:	19739:	19740:
U 18C8	19741:	19743:	19746:	19747:	19749:	19750:	19752:	19755:
U 18D0	19756:	19757:	19758:	19759:	19760:	19763:	19764:	19765:
U 18D8	19766:	19768:	19769:	19770:	19771:	19773:	19774:	19775:
U 18E0	19776:	19777:	19779:	19780:	19781:	19783:	19784:	19785:
U 18E8	19787:	19788:	19790:	19791:	19793:	19795:	19796:	19798:
U 18F0	19799:	19801:	19802:	19803:	19804:	19805:	19806:	19809:
U 18F8	19810:	19811:	19812:	19814:	19815:	19816:	19817:	19819:
U 1900	19820:	19821:	19822:	19823:	19825:	19826:	19827:	19829:
U 1908	19830:	19831:	19832:	19834:	19835:	19836:	19838:	19839:
U 1910	19840:	19842:	19843:	19845:	19846:	19848:	19850:	19851:
U 1918	19853:	19854:	19856:	19857:	19858:	19860:	19861:	19863:
U 1920	19866:	19867:	19868:	19869:	19870:	19871:	19874:	19876:
U 1928	19877:	19878:	19879:	19881:	19882:	19883:	19884:	19886:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 1930	19887:	19889:	19890:	19891:	19893:	19894:	19895:	19897:
U 1938	19898:	19900:	19901:	19902:	19903:	19906:	19908:	19909:
U 1940	19910:	19911:	19912:	19914:	19915:	19916:	19917:	19919:
U 1948	19920:	19922:	19923:	19924:	19926:	19927:	19928:	19930:
U 1950	19931:	19933:	19938:	19940:	19942:	19943:	19945:	19946:
U 1958	19947:	20174:	20176:	20179:	20181:	20184:	20186:	20187:
U 1960	20188:	20189:	20190:	20191:	20193:	20196:	20197:	20198:
U 1968	20199:	20200:	20202:	20203:	20204:	20205:	20206:	20207:
U 1970	20208:	20209:	20211:	20213:	20215:	20216:	20217:	20218:
U 1978	20219:	20220:	20222:	20224:	20225:	20227:	20228:	20230:
U 1980	20232:	20234:	20235:	20236:	20238:	20239:	20241:	20242:
U 1988	20243:	20244:	20246:	20248:	20250:	20251:	20253:	20254:
U 1990	20255:	20256:	20258:	20259:	20260:	20261:	20263:	20264:
U 1998	20265:	20266:	20267:	20268:	20269:	20270:	20271:	20273:
U 19A0	20275:	20276:	20278:	20279:	20281:	20283:	20285:	20286:
U 19A8	20287:	20289:	20290:	20292:	20293:	20294:	20295:	20297:
U 19B0	20298:	20300:	20302:	20304:	20305:	20306:	20307:	20308:
U 19B8	20310:	20311:	20312:	20313:	20314:	20318:	20320:	20321:
U 19C0	20322:	20323:	20325:	20326:	20328:	20330:	20332:	20333:
U 19C8	20334:	20335:	20336:	20337:	20339:	20340:	20342:	20343:
U 19D0	20344:	20345:	20346:	20350:	20352:	20353:	20354:	20355:
U 19D8	20357:	20358:	20360:	20362:	20364:	20365:	20366:	20367:
U 19E0	20368:	20370:	20371:	20373:	20374:	20376:	20377:	20378:
U 19E8	20379:	20381:	20382:	20383:	20384:	20385:	20387:	20390:
U 19F0	20393:	20394:	20397:	20400:	20401:	20404:	20407:	20408:
U 19F8	20411:	20414:	20415:	20418:	20421:	20423:	20424:	20425:
U 1A00	20426:	20428:	20429:	20431:	20432:	20433:	20434:	20435:
U 1A08	20436:	20437:	20439:	20442:	20444:	20445:	20446:	20447:
U 1A10	20449:	20451:	20453:	20455:	20456:	20457:	20459:	20461:
U 1A18	20462:	20463:	20466:	20469:	20470:	20471:	20472:	20473:
U 1A20	20475:	20476:	20478:	20479:	20482:	20483:	20484:	20486:
U 1A28	20487:	20488:	20489:	20491:	20492:	20494:	20495:	20496:
U 1A30	20497:	20500:	20503:	20504:	20505:	20506:	20507:	20509:
U 1A38	20510:	20512:	20513:	20514:	20516:	20518:	20519:	20520:
U 1A40	20522:	20523:	20524:	20525:	20526:	20528:	20529:	20530:
U 1A48	20531:	20533:	20534:	20535:	20536:	20537:	20539:	20543:
U 1A50	20544:	20545:	20546:	20547:	20549:	20553:	20554:	20555:
U 1A58	20556:	20558:	20559:	20560:	20561:	20563:	20564:	20567:
U 1A60	20568:	20571:	20572:	20573:	20575:	20576:	20577:	20578:
U 1A68	20580:	20581:	20582:	20583:	20584:	20586:	20587:	20588:
U 1A70	20589:	20591:	20592:	20595:	20596:	20598:	20599:	20601:
U 1A78	20602:	20603:	20605:	20606:	20607:	20608:	20609:	20611:
U 1A80	20612:	20613:	20614:	20616:	20617:	20619:	20620:	20621:
U 1A88	20622:	20624:	20625:	20628:	20630:	20632:	20633:	20634:
U 1A90	20635:	20637:	20638:	20640:	20649:	20650:	20651:	20652:
U 1A98	20654:	20655:	20656:	20657:	20658:	20660:	20668:	20669:
U 1AA0	20670:	20671:	20673:	20674:	20675:	20676:	20677:	20679:
U 1AA8	20680:	20681:	20682:	20683:	20685:	20686:	20687:	20688:
U 1AB0	20689:	20691:	20700:	20703:	20704:	20706:	20707:	20708:
U 1AB8	20710:	20712:	20713:	20716:	20718:	20720:	20832:	20834:
U 1AC0	20837:	20839:	20842:	20844:	20845:	20847:	20849:	20850:
U 1AC8	20854:	20856:	20857:	20861:	20862:	20864:	20868:	20869:
U 1AD0	20870:	20871:	20873:	20874:	20875:	20876:	20877:	20879:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 1AD8	20880:	20881:	20882:	20884:	20885:	20886:	20887:	20888:
U 1AE0	20889:	20892:	20893:	20894:	20895:	20898:	20899:	20900:
U 1AE8	20902:	20903:	20905:	20908:	20910:	20913:	20915:	20916:
U 1AF0	20918:	20919:	20920:	20922:	20924:	20926:	20927:	20928:
U 1AF8	20930:	20931:	20933:	20935:	20936:	20938:	20939:	20943:
U 1B00	20946:	20947:	20948:	20949:	20950:	20951:	20952:	20954:
U 1B08	20955:	20957:	20958:	20959:	20961:	20963:	20966:	20967:
U 1B10	20969:	20970:	20971:	20972:	20975:	20976:	20979:	20982:
U 1B18	20983:	20985:	20986:	20988:	20992:	20994:	20996:	21000:
U 1B20	21002:	21003:	21005:	21006:	21007:	21009:	21011:	21014:
U 1B28	21015:	21017:	21020:	21021:	21022:	21024:	21025:	21027:
U 1B30	21029:	21030:	21032:	21033:	21037:	21038:	21039:	21040:
U 1B38	21041:	21042:	21043:	21045:	21046:	21048:	21049:	21050:
U 1B40	21052:	21054:	21058:	21059:	21061:	21065:	21066:	21068:
U 1B48	21069:	21070:	21071:	21074:	21075:	21078:	21079:	21081:
U 1B50	21082:	21084:	21087:	21089:	21090:	21091:	21092:	21093:
U 1B58	21096:	21097:	21099:	21100:	21101:	21103:	21105:	21107:
U 1B60	21109:	21110:	21113:	21116:	21117:	21118:	21119:	21120:
U 1B68	21122:	21124:	21125:	21127:	21128:	21129:	21131:	21133:
U 1B70	21136:	21137:	21140:	21142:	21146:	21147:	21148:	21150:
U 1B78	21151:	21152:	21155:	21156:	21158:	21159:	21160:	21161:
U 1B80	21163:	21164:	21166:	21167:	21168:	21169:	21171:	21172:
U 1B88	21173:	21174:	21176:	21177:	21178:	21179:	21181:	21183:
U 1B90	21184:	21185:	21186:	21188:	21190:	21191:	21194:	21197:
U 1B98	21198:	21199:	21200:	21202:	21203:	21206:	21208:	21209:
U 1BA0	21211:	21212:	21213:	21214:	21216:	21217:	21219:	21220:
U 1BA8	21221:	21222:	21224:	21225:	21226:	21227:	21229:	21230:
U 1BB0	21231:	21232:	21234:	21236:	21237:	21238:	21239:	21242:
U 1BB8	21243:	21246:	21256:	21258:	21260:	21261:	21264:	21266:
U 1BC0	21273:	21274:	21275:	21277:	21279:	21280:	21282:	21283:
U 1BC8	21284:	21291:	21292:	21293:	21479:	21481:	21484:	21486:
U 1BD0	21489:	21491:	21493:	21494:	21497:	21498:	21500:	21502:
U 1BD8	21503:	21505:	21506:	21507:	21508:	21509:	21510:	21512:
U 1BE0	21513:	21514:	21516:	21517:	21519:	21520:	21521:	21522:
U 1BE8	21523:	21525:	21528:	21529:	21530:	21532:	21536:	21537:
U 1BF0	21627:	21629:	21632:	21634:	21637:	21638:	21639:	21640:
U 1BF8	21641:	21642:	21643:	21645:	21646:	21649:	21651:	21652:
U 1C00	21654:	21655:	21656:	21657:	21659:	21661:	21663:	21664:
U 1C08	21666:	21667:	21668:	21669:	21671:	21672:	21674:	21676:
U 1C10	21677:	21679:	21680:	21681:	21682:	21684:	21685:	21687:
U 1C18	21688:	21690:	21691:	21692:	21695:	21697:	21698:	21700:
U 1C20	21701:	21702:	21703:	21705:	21707:	21709:	21710:	21712:
U 1C28	21713:	21714:	21715:	21717:	21719:	21721:	21722:	21724:
U 1C30	21725:	21726:	21727:	21729:	21731:	21733:	21734:	21736:
U 1C38	21737:	21738:	21739:	21741:	21742:	21743:	21744:	21747:
U 1C40	21749:	21750:	21752:	21753:	21754:	21755:	21757:	21758:
U 1C48	21759:	21761:	21763:	21765:	21767:	21768:	21769:	21770:
U 1C50	21772:	21773:	21774:	21775:	21777:	21779:	21780:	21782:
U 1C58	21783:	21784:	21785:	21841:	21843:	21846:	21848:	21851:
U 1C60	21852:	21853:	21854:	21855:	21856:	21858:	21859:	21861:
U 1C68	21863:	21866:	21867:	21869:	21870:	21871:	21872:	21874:
U 1C70	21875:	21876:	21877:	21879:	21880:	21882:	21883:	21884:
U 1C78	21885:	22023:	22025:	22028:	22030:	22033:	22034:	22035:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 1C80	22036:	22037:	22038:	22041:	22042:	22043:	22044:	22045:
U 1C88	22046:	22048:	22049:	22050:	22051:	22052:	22053:	22054:
U 1C90	22055:	22057:	22058:	22060:	22062:	22063:	22064:	22065:
U 1C98	22066:	22067:	22069:	22070:	22071:	22073:	22074:	22075:
U 1CA0	22076:	22077:	22078:	22079:	22081:	22082:	22083:	22085:
U 1CA8	22086:	22087:	22088:	22089:	22090:	22092:	22093:	22095:
U 1CB0	22103:	22105:	22107:	22108:	22110:	22111:	22112:	22114:
U 1CB8	22115:	22117:	22119:	22120:	22121:	22122:	22123:	22125:
U 1CC0	22126:	22127:	22128:	22129:	22130:	22132:	22133:	22134:
U 1CC8	22136:	22137:	22138:	22140:	22141:	22143:	22144:	22145:
U 1CD0	22146:	22148:	22149:	22150:	22152:	22153:	22155:	22156:
U 1CD8	22158:	22159:	22160:	22161:	22162:	22163:	22167:	22168:
U 1CE0	22170:	22171:	22172:	22173:	22174:	22175:	22176:	22354:
U 1CE8	22356:	22359:	22361:	22364:	22365:	22366:	22367:	22368:
U 1CF0	22370:	22373:	22374:	22375:	22377:	22379:	22382:	22383:
U 1CF8	22384:	22385:	22387:	22389:	22390:	22391:	22392:	22395:
U 1D00	22397:	22398:	22400:	22401:	22402:	22403:	22406:	22407:
U 1D08	22408:	22409:	22411:	22412:	22413:	22414:	22417:	22418:
U 1D10	22419:	22420:	22422:	22426:	22427:	22428:	22430:	22431:
U 1D18	22432:	22433:	22434:	22436:	22442:	22444:	22446:	22448:
U 1D20	22449:	22451:	22453:	22454:	22456:	22457:	22459:	22461:
U 1D28	22463:	22465:	22466:	22467:	22470:	22471:	22473:	22475:
U 1D30	22476:	22477:	22479:	22480:	22481:	22482:	22484:	22485:
U 1D38	22486:	22487:	22489:	22490:	22491:	22492:	22493:	22495:
U 1D40	22496:	22500:	22501:	22503:	22504:	22505:	22506:	22507:
U 1D48	22508:	22509:	22708:	22710:	22713:	22715:	22718:	22719:
U 1D50	22720:	22721:	22722:	22724:	22727:	22728:	22729:	22731:
U 1D58	22732:	22734:	22736:	22738:	22739:	22741:	22742:	22744:
U 1D60	22745:	22747:	22748:	22749:	22750:	22751:	22752:	22753:
U 1D68	22756:	22758:	22759:	22761:	22762:	22763:	22764:	22765:
U 1D70	22766:	22767:	22770:	22772:	22773:	22775:	22776:	22777:
U 1D78	22778:	22779:	22780:	22781:	22784:	22786:	22787:	22789:
U 1D80	22790:	22791:	22792:	22793:	22794:	22797:	22798:	22799:
U 1D88	22800:	22802:	22806:	22807:	22808:	22810:	22811:	22813:
U 1D90	22814:	22815:	22816:	22817:	22821:	22823:	22824:	22826:
U 1D98	22827:	22829:	22830:	22831:	22832:	22833:	22834:	22835:
U 1DA0	22839:	22846:	22849:	22851:	22852:	22854:	22855:	22857:
U 1DA8	22861:	22863:	22865:	22867:	22868:	22869:	22871:	22872:
U 1DB0	22873:	22874:	22876:	22877:	22879:	22880:	22881:	22882:
U 1DB8	22884:	22886:	22887:	23020:	23022:	23025:	23027:	23030:
U 1DC0	23031:	23032:	23033:	23034:	23035:	23036:	23037:	23039:
U 1DC8	23042:	23043:	23044:	23045:	23046:	23047:	23049:	23050:
U 1DD0	23051:	23052:	23055:	23056:	23058:	23059:	23060:	23061:
U 1DD8	23062:	23063:	23064:	23065:	23066:	23069:	23070:	23071:
U 1DE0	23072:	23074:	23075:	23076:	23077:	23080:	23081:	23082:
U 1DE8	23083:	23085:	23086:	23087:	23088:	23091:	23098:	23101:
U 1DF0	23104:	23106:	23109:	23111:	23113:	23114:	23116:	23117:
U 1DF8	23119:	23123:	23125:	23127:	23129:	23130:	23131:	23133:
U 1E00	23134:	23135:	23136:	23138:	23139:	23141:	23142:	23143:
U 1E08	23144:	23146:	23148:	23149:	23255:	23257:	23260:	23262:
U 1E10	23265:	23266:	23267:	23268:	23270:	23271:	23272:	23273:
U 1E18	23274:	23276:	23278:	23281:	23282:	23283:	23284:	23286:
U 1E20	23290:	23291:	23292:	23295:	23296:	23298:	23299:	23301:

:
: Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 1E28	23302:	23304:	23305:	23306:	23307:	23308:	23311:	23312:
U 1E30	23313:	23314:	23316:	23319:	23320:	23321:	23323:	23327:
U 1E38	23328:	23330:	23331:	23333:	23334:	23335:	23336:	23337:
U 1E40	23338:	23341:	23348:	23350:	23352:	23353:	23355:	23356:
U 1E48	23358:	23362:	23364:	23366:	23368:	23369:	23370:	23372:
U 1E50	23373:	23374:	23375:	23377:	23378:	23380:	23381:	23382:
U 1E58	23383:	23385:	23387:	23388:	23587:	23589:	23592:	23594:
U 1E60	23597:	23598:	23599:	23600:	23601:	23602:	23604:	23605:
U 1E68	23606:	23607:	23608:	23611:	23612:	23613:	23615:	23618:
U 1E70	23619:	23620:	23621:	23622:	23624:	23627:	23629:	23630:
U 1E78	23631:	23633:	23635:	23637:	23638:	23639:	23641:	23642:
U 1E80	23643:	23644:	23645:	23646:	23647:	23649:	23650:	23653:
U 1E88	23654:	23655:	23657:	23658:	23659:	23660:	23661:	23662:
U 1E90	23663:	23666:	23667:	23668:	23669:	23670:	23671:	23673:
U 1E98	23676:	23678:	23679:	23680:	23682:	23683:	23684:	23685:
U 1EA0	23686:	23687:	23688:	23691:	23692:	23693:	23695:	23696:
U 1EA8	23697:	23698:	23699:	23700:	23701:	23702:	23705:	23706:
U 1EB0	23707:	23708:	23710:	23714:	23715:	23716:	23718:	23719:
U 1EB8	23721:	23722:	23723:	23724:	23725:	23726:	23727:	23728:
U 1EC0	23729:	23730:	23734:	23735:	23736:	23737:	23739:	23740:
U 1EC8	23741:	23743:	23744:	23745:	23748:	23749:	23750:	23751:
U 1ED0	23753:	23757:	23758:	23759:	23761:	23762:	23764:	23765:
U 1ED8	23766:	23769:	23773:	23774:	23775:	23776:	23777:	23779:
U 1EE0	23783:	23784:	23785:	23787:	23788:	23789:	23790:	23791:
U 1EE8	23792:	23793:	23794:	23798:	23800:	23804:	23805:	23806:
U 1EF0	23808:	23809:	23811:	23814:	23820:	23822:	23824:	23827:
U 1EF8	23829:	23830:	23832:	23833:	23835:	23839:	23841:	23843:
U 1F00	23844:	23845:	23847:	23849:	23850:	23851:	23853:	23854:
U 1F08	23855:	23857:	23858:	23860:	23861:	23862:	23863:	23865:
U 1F10	23866:	23867:	23868:	23869:	23871:	23872:	23873:	23875:
U 1F18	23877:	23878:	24007:	24009:	24012:	24014:	24017:	24018:
U 1F20	24019:	24020:	24021:	24022:	24023:	24025:	24028:	24029:
U 1F28	24030:	24032:	24033:	24035:	24037:	24040:	24041:	24043:
U 1F30	24044:	24045:	24046:	24047:	24048:	24051:	24052:	24053:
U 1F38	24054:	24055:	24058:	24059:	24060:	24061:	24063:	24067:
U 1F40	24068:	24069:	24070:	24071:	24072:	24074:	24080:	24081:
U 1F48	24083:	24085:	24086:	24088:	24089:	24091:	24094:	24097:
U 1F50	24099:	24101:	24102:	24104:	24105:	24106:	24107:	24109:
U 1F58	24275:	24277:	24280:	24282:	24285:	24286:	24287:	24288:
U 1F60	24289:	24290:	24291:	24292:	24294:	24297:	24298:	24299:
U 1F68	24300:	24301:	24302:	24304:	24305:	24306:	24307:	24310:
U 1F70	24311:	24313:	24314:	24317:	24319:	24322:	24324:	24325:
U 1F78	24328:	24329:	24331:	24332:	24335:	24337:	24338:	24339:
U 1F80	24340:	24341:	24344:	24345:	24348:	24350:	24352:	24353:
U 1F88	24354:	24357:	24358:	24361:	24363:	24365:	24366:	24367:
U 1F90	24369:	24372:	24373:	24374:	24378:	24385:	24386:	24388:
U 1F98	24390:	24392:	24393:	24395:	24396:	24398:	24402:	24404:
U 1FA0	24406:	24408:	24409:	24411:	24412:	24413:	24415:	24417:
U 1FA8	24418:	24420:	24421:	24422:	24423:	24424:	24425:	24427:
U 1FB0	24428:	24429:	24430:	24432:	24433:	24434:	24435:	24437:
U 1FB8	24439:	24440:	24444:	24446:	24447:	24449:	24451:	24452:
U 1FC0	24454:	24455:	24457:	24461:	24463:	24465:	24466:	24467:
U 1FC8	24469:	24471:	24473:	24474:	24477:	24478:	24480:	24481:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 1FD0	24483:	24484:	24485:	24486:	24487:	24488:	24490:	24491:
U 1FD8	24492:	24493:	24495:	24496:	24497:	24498:	24500:	24502:
U 1FE0	24503:	24601:	24603:	24606:	24608:	24610:		
U 1FE8 - 1FFF Unused								
U 2000	24614:	24615:	24616:	24617:	24619:	24620:	24621:	24622:
U 2008	24623:	24625:	24626:	24627:	24629:	24633:	24634:	24635:
U 2010	24636:	24637:	24639:	24643:	24645:	24646:	24648:	24650:
U 2018	24651:	24652:	24655:	24656:	24657:	24659:	24660:	24661:
U 2020	24662:	24663:	24664:	24666:	24669:	24671:	24672:	24673:
U 2028	24676:	24677:	24678:	24679:	24681:	24685:	24686:	24687:
U 2030	24688:	24689:	24690:	24691:	24693:	24696:	24698:	24699:
U 2038	24700:	24703:	24709:	24711:	24712:	24714:	24716:	24717:
U 2040	24719:	24720:	24722:	24726:	24728:	24730:	24731:	24732:
U 2048	24734:	24736:	24737:	24738:	24740:	24741:	24742:	24744:
U 2050	24745:	24746:	24747:	24748:	24750:	24751:	24752:	24754:
U 2058	24756:	24757:	24798:	24800:	24803:	24805:	24808:	24809:
U 2060	24810:	24811:	24812:	24813:	24814:	24815:	24817:	24819:
U 2068	24820:	24822:	24823:	24824:	24825:	24888:	24890:	24893:
U 2070	24895:	24898:	24899:	24900:	24901:	24902:	24904:	24907:
U 2078	24908:	24909:	24911:	24913:	24915:	24917:	24919:	24921:
U 2080	24923:	24926:	24927:	24930:	24932:	24933:	24935:	24936:
U 2088	24938:	24941:	24943:	24945:	24947:	24948:	24949:	24951:
U 2090	24952:	24953:	24954:	25078:	25080:	25083:	25085:	25088:
U 2098	25089:	25090:	25091:	25092:	25094:	25098:	25099:	25101:
U 20A0	25103:	25104:	25105:	25107:	25109:	25111:	25113:	25114:
U 20A8	25115:	25117:	25119:	25120:	25123:	25125:	25132:	25133:
U 20B0	25134:	25135:	25136:	25139:	25141:	25143:	25144:	25146:
U 20B8	25148:	25150:	25153:	25155:	25158:	25159:	25160:	25162:
U 20C0	25163:	25165:	25166:	25170:	25171:	25173:	25175:	25176:
U 20C8	25177:	25178:	25180:	25181:	25182:	25183:	25186:	25187:
U 20D0	25189:	25190:	25192:	25193:	25194:	25195:	25196:	25198:
U 20D8	25199:	25201:	25202:	25205:	25206:	25207:	25208:	25210:
U 20E0	25211:	25212:	25213:	25217:	25218:	25220:	25222:	25223:
U 20E8	25224:	25225:	25227:	25228:	25229:	25230:	25233:	25235:
U 20F0	25236:	25240:	25241:	25243:	25245:	25246:	25248:	25251:
U 20F8	25252:	25254:	25257:	25258:	25260:	25262:	25263:	25266:
U 2100	25268:	25269:	25270:	25271:	25275:	25277:	25279:	25280:
U 2108	25283:	25285:	25286:	25287:	25290:	25291:	25292:	25294:
U 2110	25295:	25297:	25298:	25301:	25303:	25304:	25305:	25367:
U 2118	25369:	25372:	25374:	25377:	25378:	25379:	25380:	25381:
U 2120	25383:	25386:	25387:	25389:	25392:	25393:	25394:	25395:
U 2128	25397:	25400:	25403:	25405:	25406:	25407:	25409:	25410:
U 2130	25412:	25414:	25415:	25416:	25417:	25419:	25421:	25422:
U 2138	25423:	25424:	25426:	25433:	25434:	25436:	25438:	25439:
U 2140	25441:	25443:	25445:	25448:	25450:	25453:	25454:	25455:
U 2148	25457:	25459:	25460:	25461:	25462:	25464:	25466:	25467:
U 2150	25468:	25469:	25471:	25472:	25473:	25474:	25476:	25477:
U 2158	25479:	25482:	25483:	25486:	25507:	25509:	25512:	25514:
U 2160	25517:	25518:	25519:	25520:	25521:	25523:	25524:	25525:
U 2168	25526:	25527:	25528:	25529:	25530:	25531:	25532:	25533:
U 2170	25536:	25537:	25538:	25539:	25540:	25541:	25542:	25543:
U 2178	25544:	25548:	25549:	25552:	25554:	25562:	25564:	25566:
U 2180	25568:	25571:	25574:	25575:	25577:	25578:	25579:	25581:

Location / Line Number Index

;Location	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
U 2188	25582:	25584:	25586:	25587:	25590:	25591:	25592:	25594:
U 2190	25595:	25597:	25599:	25600:	25608:	25610:	25611:	25613:
U 2198	25615:	25616:	25617:	25618:	25619:	25622:	25623:	25624:
U 21A0	25626:	25628:	25629:	25631:	25633:	25634:	25635:	25637:
U 21A8	25640:	25647:	25650:	25651:	25652:	25654:	25655:	25656:
U 21B0	25657:	25658:	25659:	25662:	25669:	25670:	25671:	25672:
U 21B8	25679:	25680:	25681:	25682:	25685:	25687:	25688:	25689:
U 21C0	25693:	25694:	25695:	25696:	25697:	25699:	25701:	25705:
U 21C8	25706:	25707:	25708:	25710:	25712:	25716:	25717:	25718:
U 21D0	25720:	25721:	25722:	25724:	25725:	25727:	25730:	25731:
U 21D8	25733:	25738:	25740:	25741:	25742:	25743:	25745:	25746:
U 21E0	25747:	25748:	25750:	25753:	25756:			

C-code Microword Summary

	Words not in bounds
ENKCC	512
Used	512
Remaining	

Total microwords used in memory C: 512
Total microwords remaining in memory C: 0
Highest address used in memory C: 180 (hex)

U-code Microword Summary

	Words not in bounds
ENKCC	7681
Used	7681
Remaining	

Total microwords used in memory U: 7681
Total microwords remaining in memory U: 0
Highest address used in memory U: 21E4 (hex)

; The file used in this assembly is:
ENKCC.MIC

Pass 1 warnings: 0 Pass 2 warnings: 0
Pass 1 errors: 0 Pass 2 errors: 0