16. HARDWARE DIAGNOSTIC and MAINTENANCE PROCEDURES

This section discusses operation of various hardware diagnostic software and firmware. Microprocessor diagnostics are normally used to isolate suspected failures of the microprocessor or port hardware. Micro-Exec memory and disk testing commands are generally used to isolate suspected failures of disk units and main storage modules.

The microprocessor diagnostics require that a small nucleus of the microprocessor and its interface to the Nova/Alto be working, before meaningful failure diagnosis can take place. When a failure has occurred in this essential nucleus, the Midas "Test-All" command (*Maxc2 only*) and SMIDiag.run (*Maxc2 only*) are used to isolate the failure. TR has been used for this purpose on Maxc1, but the last year or so we have not been able to recover TR from any of the old tapes on which it is stored. We have had to patch test loops into NVIO and use a scope to fix problems in the essential nucleus on Maxc1.

MemBash and TM (*Maxc2 only*) have been used to check out memory reference interference problems and failures in the Alto memory interface.

16.1. Running Microprocessor Diagnostics

The microprocessor diagnostics are normally kept on the Nova or Alto disk and (for Maxc1) on the 10SYS tape.

You must obtain the notebook labelled "Microprocessor Diagnostics" or "Maxc Diagnostics" in order to do any useful debugging. However, the diagnostics all fall into a common pattern and they can usually be run in a simple-minded way without understanding too much about what the programs are really doing. The following generalizations may be helpful.

Associated with each of the diagnostics described below is a command file whose name is the same as that of the diagnostic. If you type to Nova DOS or to the Alto Executive:

MIDAS diagname <cr>

Midas will load the diagnostic and show registers pertinent to the diagnostic's operation on the display. You may wish to set some parameters before starting a diagnostic (for example, the low and high addresses for a memory test). However, the values as loaded are reasonable for thorough testing.

When you are ready, type START;G to start the diagnostic. (DGM has two alternate starting addresses for interrupt system testing.) The diagnostic will halt after one pass at IMA=20 (DGIML is the only exception to this rule: it breaks at 3200 on Maxc1 and 7200 on Maxc2). This means that no failures were detected. To repeat, type ;P to Midas. To loop indefinitely, delete the breakpoint at 20 by typing 20;K and then ;P. All of the diagnostics except DGM loop in less than two seconds, so something is wrong if the machine hangs in the loop.

A breakpoint at any location except 20 indicates that an error was detected. The most common error breakpoint is at 25 or 26, the comparison error breakpoint. The location in the diagnostic

from which the comparison routine was called is displayed at STK 0. On Maxc1 the .LS listing for each diagnostic gives address symbols and their values, sorted in order of value so that the symbol nearest the error address may be found readily (you normally are interested in IM memory addresses only). On Maxc2, you can get Midas to tell you the nearest label by selecting the value to the right of "STK 0" with the middle mouse button. Following the .LS listing is the diagnostic listing, which contains general operating instructions in comments preceding the program itself. DBEG0 is assembled or loaded ahead of some diagnostics, so its listing and .LS file may also be relevant. Some diagnostics INSERT other files or are assembled from several sources, so you may have to look a bit to find the relevant listings. When tracing an error, find the tag nearest the address in STK 0 (if the breakpoint is at IMA=26) or IMA (if the breakpoint is elsewhere) and read the comments there from the listing.

If the breakpoint is at IMA=26, the diagnostic may be resumed from the point of error by the command ;P, which will return to the caller of the compare routine (this usually works but not always).

One common problem that occurs on Maxc1 when running diagnostics is that the microprocessor single steps and refuses to run. The usual cause of this is a memory interface hangup. This may be cured by performing the "Reset Memory" operation of NVIO, as follows:

```
!NVIO.SV/H/R <cr>
NVIO
:M...OK .
```

If the machine still hangs, run POWER ON and see if that cures the problem. If that fails, run POWER OFF, wait for several seconds, and run POWER ON again.

The diagnostic names and what they test are listed below:

DGBASIC.MB	Most basic diagnostic. Does not use the main memory, the interrupt system or the P-register input multiplexors. Tests everthing thing except the SM, DM, DM1, DM2, LM, and RM memories first, then tests these memories and the F-register. Cycle time < 1 second.
DGP.MB	Tests the P-register inputs and a few afterthoughts from DGBASIC.MB. Cycle time < 1 second.
DGALU.MB	Tests an assortment of P-register, Q-register, and ALU operations using random numbers. Cycle time < 1 second.
DGM.MB	Tests the memory interfaces. There are alternate entry points at START and XSTART for interrupt system testing. Be sure to read the listing comments before starting at XSTART. Cycle time ~ 20 minutes to test each 128K of main memory; correspondingly less if the address range is restricted. *Note:* After running DGM, you have to go through the "Power Up" procedure before running the PDP-10 microcode again, because certain low core locations that must be zero are not left zero by DGM. This will cause an immediate NVIO punt, if you forget to do this.

DGI.MB	Tests the interrupt system and repeats parts of DGBASIC and DGP affected by interrupts. Cycle time < 1 second.
DGIML/DGIMH.MB	Tests the SM, DM, DM1, DM2, and MP memories using random numbers, then the instruction memory (IM) first by using each of the 72 patterns of a single 0 in a field of 1's, then the 72 patterns of a single 1 in a field of 0's, then random numbers. DGIML runs in the top of IM and tests the memory below it, while DGIMH runs in the bottom of IM and tests the memory above it. Cycle time < 3 seconds. Alternate starting addresses exist to test only IM, only MP, or only SM/DM/DM1/DM2 (which are physically a single memory). There are a number of special loops for repeating test sequences that have provoked failures in the past.
DGRL.MB	Tests the right register bank (RM) and the left register bank (LM) using random numbers. Cycle time < 1 second.
DGMR.MB	Tests the main memory using random numbers. Cycle time ~ 5 seconds. *Note:* After running DGMR, you have to go through the "Power Up" procedure before running the PDP-10 microcode to clear several low core locations that must be zeroed but are smashed by DGMR.
DGREG.MB	Tests assorted registers with random numbers. Like DGALU, DGREG is a reliability diagnostic that supplements the basic tests in DGBASIC and DGP. Cycle time < 1 second.

16.2. Running PDP-10 Diagnostics

PDP-10 instruction diagnostics 0A through 0N and 0R may be run on Maxc either in stand-alone mode or under Tenex. We use them only for checking out new PDP-10 emulator microcode and not for hardware diagnosis (which is what they were originally intended for). Some of them have been patched to account for Maxc incompatibilities.

There are two methods of running PDP-10 diagnostics stand-alone. The more convenient is to run them from Micro-Exec by means of the "run.diagnostic.program" command. However, if the microcode is working so poorly that Micro-Exec won't run, the other method is to load them from the Nova disk using DMPLD or from the Alto using AltIO's "Load" command.

Running diagnostics from Micro-Exec is simple. All the diagnostics that will run on Maxc are stored as programs on save area 2. They may be listed out by the "Print.Program.Directory" command. To execute a single pass of a given diagnostic, type:

*Run.Diagnostic.Program <program name> <cr>

Control returns to Micro-Exec when the diagnostic is finished. To make the program loop forever, type:

*Goto 4000 <cr>

The iteration count (a decrementing negative number) may be displayed by examining Maxc location 1. To abort this, you have to either reboot Micro-Exec or halt the microprocessor with NVIO/AltIO "H" command and then restart Micro-Exec with "20G."

Running diagnostics using DMPLD or AltIO is somewhat messier. This procedure should be used only when, due to hardware or microcode problems, it is impossible to run the diagnostics from Micro-Exec.

Maxc1: The procedure is as follows:

1) Load the diagnostics from the "PDP-10 Diagnostics" tape using the DOS "Load" command. (It is not possible simply to FTP them from Maxc2 because the file format used by DMPLD is different from that used by FTP.)

2) Load the PDP-10 microcode by means of the "MIDAS TENLOAD" command.

3) Load the selected diagnostic into Maxc memory by means of DMPLD (see Section 15).

4) Enter NVIO by typing:

!NVIO.SV/H <cr>

5) Using ODT, make the patches listed on a sheet of paper at the beginning of the diagnostic listing.

6) Start the diagnostic by typing:

:4000G...OK .

Maxc2: The procedure is as follows:

1) Retrieve the diagnostics from the <DIAGNOSTICS> directory on Maxc1 using FTP. Only a few diagnostics will fit on the Alto disk at once.

2) Load the PDP-10 microcode by means of the "MIDAS TENLOAD" command.

3) Enter AltIO by selecting the "AltIO", "Dont-Go", and "Do-It" menu items.

4) Load the selected diagnostic into Maxc memory by means of AltIO's "Load" command. Then make sure all the patches have been made.

5) Start the diagnostic by issuing the "Go" command.

On Maxc1, the <DIAGNOSTICS> directory contains all the PDP-10 diagnostics, along with a set of RUNFILs for running them in user mode under Tenex (note that diagnostic 0D cannot be run in user mode). The script named <diagnostic name>.RUNFIL will cause the specified diagnostic to be started and run forever (type control-B to stop). Additionally, there are command files BASIC.RUNFIL and RELIABILITY.RUNFIL which execute one pass of diagnostics 0A-0H and 0I-0N respectively.

16.3. Memory Maintenance

Memory maintenance is scheduled periodically every 3 months or so to replace storage ic's that have failed. Because the memory is error-corrected, the system can be operated with a number of bad components, so memory maintenance is not scheduled until the number of failures becomes significant.

The procedure for doing this is as follows:

1) Schedule the system down using the procedure discussed in "Stopping Tenex."

2) At the appointed time, the system will halt; then unprotect NVIO/AltIO with 3301P.

3) Boot MicroExec with the NVIO/AltIO "B" command.

4) Run "Test.Memory.Slow" as discussed below. You want to get output on paper; on Maxc1, the console terminal has paper output, so nothing special is required; on Maxc2, you should issue the "Diablo Printer On" command to AltIO before starting the test.

5) Power down the system as discussed in the "Power Down" section. On Maxc2, it is possible to power-off only the memory (so you don't have to turn off the disks).

6) Pull the cards affected by bad chips and mark the bad ic's with a magic marker; interpret the output of "Test.Memory.Slow" as discussed below. Record the serial numbers of the boards pulled from each position and attempt to restore the cards to their original positions after repair.

7) Replace the bad 1103 ic's or, if that isn't possible, use the spare memory cards in the rack above the Maxc2 Alto. (Refer to ***Appendix A*** for diagrams showing memory board and chip locations.)

8) Restore the repaired cards to their original positions (if using a spare card, mark the bad card with the complete failure information using a piece of paper and scotch tape).

9) Power up the system as discussed in the "Power Up" section.

10) Reload the microcode and restart AltIO/NVIO and MicroExec using "Midas MExecGo," as discussed in the "Loading the PDP-10 Emulator" section; repeat Test.Memory.Slow to see if the repairs have been successful.

11) If everything is ok, restart Tenex.

Memory maintenance is performed with the aid of Micro-Exec.[1] After loading the microcode and starting up Micro-Exec, simply type:

*Test.Memory.Slow <cr>

This requires about 6 minutes to test all 384K of memory. All data, tag, and parity bit failures cause an error count to be accumulated for the affected chip. The physical location of each chip with errors is reported at the end of testing each memory cabinet, along with the pages affected and the error count. Every word in memory is tested 82 times using various patterns, so every chip is written into and read from 83968 times. Hence, a total chip failure will generate on the order of 40000 errors (since it will yield the wrong value approximately half the time), while a solid single bit failure will generate on the order of 40 errors.

Conclusively diagnosed check bit failures are also reported in this manner. For check bit failures that Micro-Exec cannot diagnose (due to an insufficient sample or lack of any obviously failing bit pattern), information is printed out as to the frequency of zeroes and ones in the *correct* values of each check bit. Since the check bits can't actually be read, the best that can be done is to deduce which check bit is failing on the basis of these frequencies.

The "Test.Memory.Verbose" command does everything that "Test.Memory.Slow" does, but also prints out the address, error type, correct data, incorrect data, and exclusive or for every error. For check bit failures, the correct data and the computed Hamming code are printed out. This command will generate reams of output unless the memory is pretty clean to begin with.

If the diagnostic crashes due to "fatal error from wrong place in code", one should re-boot Micro-Exec and issue the "Test.Memory.Write.Slow" command before running the memory test. This sets a mode switch that forces Micro-Exec to use a more conservative (but slower) method of writing data into the memory under test.

Maxc2: To obtain hardcopy of the memory error printout, issue the "Diablo Copy On" command in the AltIO command window before starting the memory test.

Because of error correction, Tenex can run ok with bad bits and even bad cards in the memory system. Exception: a double error in pages 0 to about 217 will prevent Tenex from running. Non-hardware-maintainers should ordinarily not attempt to repair the memory system. However, the following instructions for locating failures reported by Test.Memory.Slow (or Test.Memory.Fast) are provided, just in case.

The error printout is of the form:

Quad q Cab c Card d Col h Row r Pages 1360-1367 bit 14, 10 errors

where q is J, K, L, or M; c is 0 to 3; d is 1 to 16; h is 0 to 11; and r is 0 to 7.

---

[1] There is also a Maxc memory diagnostic called TMEM which runs on the Nova, but it does not test memory as thoroughly, does not find bad check bits, and requires the use of a second program, PER, to analyze failures. These programs are therefore not documented here.

The cabinets are numbered consecutively 0, 1, 2, ... starting with the one next to the processor. The quadrants are each represented by a row of 16 cards starting at the top of the cabinet, so J is the top row, then K, then L, and M is the bottom row. In each row the cards are numbered 1 to 16 starting at the left as you look from the back of the cabinet. (Ref. **Figure 4, *Appendix.***)

If you hold the cards with the ic's up, edge connector toward you, the storage chips form an array of 12 columns by 8 rows starting with 0,,0 in the lower right corner. The other parts on the board are 12 address drivers on the left and right and 6 sense amps and other stuff next to the edge connector. (Ref. **Figure 5, *Appendix.***)

In an emergency, you can replace a card with one of the spares in the unused rack above the Maxc2 Alto. If you have to do this, be sure to attach the error printout and a note about where the card came from to the card you remove; leave the card on the table in the Maxc room so system maintainers can find it.

16.4. Disk Maintenance

Micro-Exec is used to diagnose most disk problems. The assortment of commands for doing this are discussed in the Micro-Exec section. Two microdiagnostic programs called DSKD and EXAM are also available. However, these programs are only of interest to Ed McCreight--they are not intended for operation by novices.

Micro-Exec reports disk addresses in the following format:

0FFPPPHHCCCS

where:

FF = function (04 = read, 10 = write)
PPP = pack number
HH = head number
CCC = cylinder number
S = sector number

Basic test procedures are as follows:

a) Scanning a system pack for errors:

*Scan.Disk.Pack.For.Errors <pack>

This prints out both soft and hard errors, and provides no way of telling which errors are hard. To find only hard errors:

*Set.Disk.Error.Retry.Count 20
*Scan.Disk.Pack.For.Errors <pack>

b) Writing and verifying a test pack: (Note that pack 100 is reserved exclusively for this purpose.)

```
*Mount.Auxiliary.Pack <drive><pack>
*Write.Disk.Test.Pattern      (Writes and verifies test pattern)
*Verify.Disk.Test.Pattern     (Verifies already written test pattern)
*Dismount.Auxiliary.Pack      (when done)
```

c) Observing disk data on a scope:

```
*Loop.On.Specified.Disk.Page <pack><cylinder><head><sector>
```

This repeatedly reads a single page, ringing the bell (Maxc1) or blinking the screen (Maxc2) whenever an error is detected. Type DEL to terminate.

Following Tenex file system crashes, where fatal read errors in directories have occurred, the following general methods help to isolate the cause of the failure.

1. Turn on the "Read-Only" switches for all the system disk packs.

2. Use Micro-Exec S.D.P.F.E to find out which drives/packs/controllers are causing trouble. If a particular pack suffers many failures, but the others seem ok, it is likely that the common controller is ok, but that something is wrong with either the unit controller or disk drive that is experiencing the failures. If the failures are confined to a particular head, it is likely that that head is misaligned or broken.

3. If a pack produces many irrecoverable read errors on one drive, try mounting it on another drive and using S.D.P.F.E. If it can be read ok on the other drive, then it is likely that there is some failure in the read electronics of the original unit controller or disk drive; otherwise, it is likely that there is a failure in the write electronics of the original controller/drive.

4. Try writing a test pattern on a test disk pack on a good drive (Pack 100 usually used for this purpose because it does not have any bad spots.). Then mount the pack on the suspect drive and use S.D.P.F.E to see if it can be read correctly. If it cannot be read correctly, you have further indication of a read electronics failure; if it can be read correctly, then you have further confirmation of a write electronics or head failure.

5. If you determine that a particular drive/controller combination is broken, you can determine whether it is the drive or controller by recabling the suspect controller to a working drive and the suspect drive to a working controller. *(Refer to last two paragraphs in section* **14.1** *for a discussion of what to do if the disk configuration is changed.)*

16.5. TM

Maxc2 only. TM is a Maxc memory diagnostic that runs on the Alto. It is used for basic debugging of the memory system, or when the memory is too sick to support Maxc programs such

as Micro-Exec. TM is documented separately in a memo entitled "The Maxc2 Memory Test Program and Other Folklore", by Larry Clark.

16.6. MemBash

Maxc2 only. MemBash is an Alto program that beats on the Alto/Maxc memory interface while monitoring the state of the Maxc processor. It is intended to be run at the same time as a Maxc main memory micro-diagnostic (DGM or DGMR) to provoke problems that arise under conditions of memory contention.

The Maxc micro-diagnostic should first be loaded using Midas and the end-of-test breakpoint at 20 removed by typing "20;K". Then exit Midas, start up MemBash, and issue the "Go" command. The program runs until any key is struck.

If any error occurs (in the processor, memory, or Alto memory interface), the state of the memory system and all the memory interfaces is displayed. The processor is then restarted at the beginning of the micro-diagnostic.

The Alto memory operations executed by MemBash are ordinarily sequential reads through the first 256K of memory. The "Alto Operation" command may be used to select write or RMW operations or to specify that the Alto beat on a single memory location.

MemBash may be exited by means of the "Quit" command.

16.7. SMIDiag

Maxc2 only. This program is a diagnostic for the Alto System Maintenance Interface (SMI). Its operation is very simple, and one should step through the various available commands using the "?" feature.

There are two main tests, one for the address register and the other for the data register. In either test, one may use data patterns consisting of all zeroes, all ones, alternating ones and zeros, or random data. *Important:* Before running the address test with random data, it is necessary to disconnect the SMI cable to Maxc and install a terminator on the Alto interface.

16.8. AITest

Maxc2 only. Alto-IMP interface test.

16.9. TR

Maxc1 only. TR is a Nova program that tests the registers and memories of the Maxc1 processor.

Since it uses the same routines as Midas to read and set the processor state, Midas will probably not run if TR doesn't run.

The program runs under DOS. It takes six kinds of arguments:

W The registers and memories are referenced by numbers as follows:

0	PC	14	--
1	IMA	15	LM
2	P	16	--
3	Q	17	RM
4	X	18	--
5	Y	19	SM
6	AC	20	--
7	F	21	DM
8	MAR	22	--
9	KMAR	23	MAP (0 to 511 only)
10	MDR	24	--
11	MDRL	25	IM[0:35]
12	KMDR	26	--
13	ARM	27	STACK
		28	--
		29	IM[36:72]
		30	--
		31	MAIN (0 to 64K-1 only)

W > 18 refers to a memory and will be written M below.

V Value, which is a positive integer < 2**48 and is taken as decimal unless suffixed by "R8". To add n zeroes to the end of the number suffix it with "En". Thus 64 = 100R8 = 1E2R8.

A Address, which is exactly like a value.

I Increment of the form i j k, where i, j, and k are Vs. The increment i j k is t for t:= i step j until k. Hence the increment 1 3 10 produces the sequence 1, 4, 7, 10.

R Rotate, of the form i j k as above, which produces the sequence t for t := i, 2 t + J while t <= k. Hence the R's.

S Sequence of addresses, which is exactly like an increment.

The simplest calls of TR are:

TR W V to write V into register W and read it back.
TR M V A to write V into address A of M and read it back.

Only errors are printed (in octal). The switch /T prints each value written and read, and /N suppresses printing. The switch /F sends the printing to a file which is given as the first argument.

Thus:

```
TR/F FOO 3 77R8
```

tests Q with 77 and writes the errors to FOO (which must not exist already).

```
TR/I W I
TR/I M I A
TR/R W R
TR/R M R A
```

test the register or memory location with each value of the increment in turn or with each value of the rotate in turn.

Two switches make sense for memories only:

```
TR/S M V S
```

tests each address in the sequence. V may be replaced by I or R if the appropriate switch is used. Each location is tested with all the values of the I or R before going on to the next.

```
TR/A M V S
```

writes V into all the addresses and then reads them back. Again I or R may be used, in which case successive values are written into successive addresses. On successive cycles of the test the i and k of s are incremented by 1.

Thus TR/A/R 19 1 0 7 100R8 1 104R8 does:

```
100  1
101  3
102  7
103  1
104  3
101  1
102  3
103  7
104  1
---
```

Summary of flags:

A	address range	TR/A/R 10 0 1 77R8 100 1 200
F	write on file	TR/F FOO 3 77R8
I	increment for value	TR 3 0 1 77R8
N	no typing	
R	rotate	TR/R 3 0 1 1E12R8
S	sequence addresses	TR/S 10 77R8 100 1 200
T	type everything	