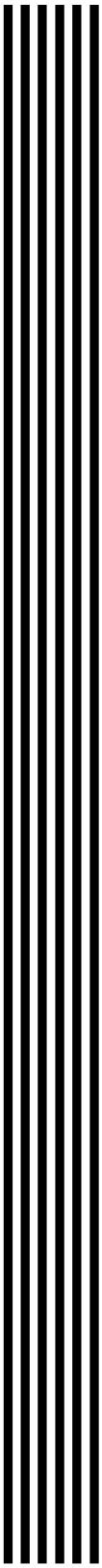

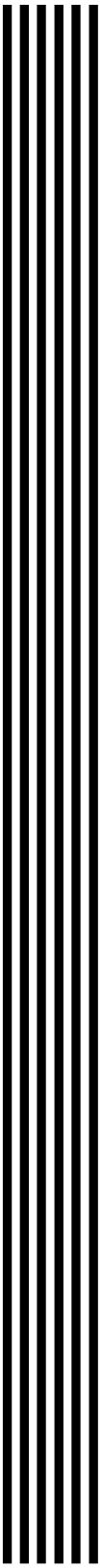

**Printer Dover**

**Spruce version 11.2 -- spooler version 11.2**

**File: MaxcOperations0-24.Press**

**Creation date: 2-Feb-81 9:45:34 PST**

**For: SIL**

**110 total sheets = 109 pages, 1 copy.**

**Problems encountered:**
**Font HELVETICA24B substituted for font HELVE¢24.**

# Maxc Operations

by Edward R. Fiala, Charles M. Geschke, and Edward Taft

Maxc Document 18.7
January 30, 1981

This document describes many of the commonly used procedures for Maxc operation, as well as a number of uncommon procedures used during system debugging and maintenance. This is intended primarily as a reference document for system personnel. However, in the absence of system personnel, any user should be able to restart Maxc from a Tenex crash using the procedure outlined in Section 2.

XEROX
PALO ALTO RESEARCH CENTER
3333 Coyote Hill Road / Palo Alto / California 94304

# 1. INTRODUCTION

This document describes many of the commonly used procedures for Maxc operation, as well as a number of uncommon procedures used during system debugging and maintenance. Also included are the complete operating instructions for several pieces of software written at PARC and unique to Maxc; these include Midas, NVIO, AltIO, DMPLD, and Micro-Exec.

"Maxc Operations" is intended primarily as a reference document for system personnel. However, in the absence of system personnel, any user should be able to restart Maxc from a Tenex crash using the procedure outlined in Section 2.

Conventions used in this document: In examples in which user typein is intermixed with machine typeout, the user typein is underlined. <cr>, <lf>, <sp>, and <esc> are used to stand for the carriage return, line feed, space, and escape keys.

All information is applicable to both Maxc1 and Maxc2 except where otherwise noted.

## 1.1 Overview of the Maxc System

From a user terminal, Maxc looks like a PDP-10 running the Tenex operating system. However, this is mostly an illusion. Maxc is really a PARC-constructed microprocessor which emulates the complete user-mode PDP-10 instruction set, as well as some privileged instructions, equivalents for the BBN pager operations, and an additional instruction set for Interlisp. The Tenex system has been considerably modified to account for the many differences between Maxc and a PDP-10, particularly in the area of input/output.

The system consists of the following major components:

A. The Maxc microprocessor itself. Besides performing PDP-10 emulation, Maxc also directly controls the disks.

B. A 384K-word by 48-bit MOS memory system, with hardware built in for correcting single-bit errors in any word and detecting (though not correcting) double errors.

C. Up to eight disk drives, each connected to its own controller inside Maxc.

D. The Maxc1 system includes a Data General Nova, which has all the other Maxc peripherals connected to it, including a data line scanner (DLS), two magtape controllers with a total of three drives (all 9-track), the IMP interface supporting Maxc's connection to the ARPA Network, and two MCA interfaces and one Ethernet interface supporting connections to all Novas and Altos at PARC. The Maxc2 system's I/O controller is an Alto, which provides Maxc access to the Alto's own monitor, a Diablo printer, and the Ethernet. All other devices are accessed through the Ethernet.

The Nova or Alto also performs a number of other important functions relating to system initialization and debugging. It has direct access to Maxc's main memory (through a separate

memory port) and indirect access to all of the internal microprocessor registers and memories. It can load microcode into Maxc's writable control store, and can command Maxc to start and stop execution, single step, and perform a number of other useful operations.

The Maxc system operates under the control of a large complex of support software, most of which is described in this document. These programs are described here, in approximately the order they would be used if one were to bring up the system from scratch.

When first booted from switches, Maxc1's Nova runs under the control of DOS, the Data General Disk Operating System, which is described in the pertinent Data General documentation. The Nova has its own Diablo moving-head disk, used for storage of Nova system programs and other important files such as the Maxc microcode.[1] DOS is also able to perform I/O to magtape unit 0 (the one directly over the Nova itself).

Assuming the necessary files have been loaded onto the Nova disk, the next program to be run is Midas. Midas is the loader and debugger for Maxc microprograms. It has a large number of commands for examining and changing microprocessor registers and memories and for starting, stopping, and single-stepping the microprocessor. Most of its capabilities are used only during debugging; in the normal course of events, it is used only to load the PDPb-10 microcode into Maxc, and it then passes control to NVIO.

NVIO is the program in control of the Nova while Tenex is running on Maxc, and contains all the necessary I/O drivers for communicating between Maxc and the peripherals connected to the Nova. NVIO also has a command decoder, by means of which the operator may examine and change memory, start and stop the Maxc microprocessor, and perform a number of other operations.

One important NVIO operation is to boot into Maxc main memory and start a PDPb-10 program called Micro-Exec. Micro-Exec has commands for performing a large variety of stand-alone Maxc operations, such as setting up the disk configuration, copying disk packs, and (most important) loading Tenex from disk and starting it.

The system software organization for Maxc2 is very similar. The Alto runs the standard Alto operating system and has a number of the usual Alto subsystems on its disk. An Alto version of Midas exists that functions similarly to the Maxc1 Nova version, but whose user interface is somewhat different. The Alto equivalent of NVIO is called AltIO. Micro-Exec and Tenex are the same for both systems, since they run in Maxc itself rather than in its I/O controller.

Most of the rest of this document is devoted to explaining the above operations in much more detail.

### 1.2. A Word on Terminals and Consoles

*Maxc1:* The Nova's controlling terminal is an Infoton display. The Maxc controlling terminal (used by Tenex and by stand-alone programs such as Micro-Exec) is the TI-700 terminal to the right of the Infoton. The Maxc log is another TI-700 off in a corner.

---

[1] This disk is not used during normal Tenex operation.

*Maxc2:* The Alto's controlling terminal is, of course, the monitor labelled "Maxc2 Alto". When the Alto is running the control program AltIO, the display is split into two main windows. The upper window is the command window for AltIO itself, whereas the lower window is the emulated Maxc controlling terminal, which serves the same purpose as the Maxc1 console TI-700. Keyboard input is directed to the window in which the blinking cursor appears; this may be changed by striking the middle or bottom unmarked key on the right side of the keyboard. The Maxc log is a Diablo printer.

*Note:* Subsequent sections discuss commands that you may have to type to various systems. If you screw up, the following table gives standard conventions for deleting the previous character or previous command typed:

| | | Delete Char | Delete Command |
|---|---|---|---|
| *Maxc1* | DOS | RUBOUT | $A^C$ |
| | Midas | RUBOUT | none |
| | NVIO | none | RUBOUT |
| *Maxc2* | Executive | BS or $A^C$ | DEL |
| | Midas | BS or $A^C$ | DEL |
| | AltIO | BS or $A^C$ | DEL |

The other TI-700 terminals in the room are connected to the Alto DLS, which is not directly a part of either Maxc system.

## 2. TENEX CRASHES

If a Maxc-Tenex maintainer is available at PARC, inform him of the crash and normally he will take over. You should try the office numbers of the maintainers, even at odd hours, because they work at irregular hours. Otherwise, be brave and read on.

If you have not already done so, you should first familiarize yourself with the material in the Introduction to this manual (Section 1). There is a map of the machine room posted on the bulletin board, and most of the equipment is labelled.

There are obvious problems in attempting to describe what to do when a system crashes. This section simply outlines a few simple procedures whose purposes are twofold: (1) to collect data about the crash for subsequent analysis and (2) to restart the machine quickly, with as little state lost as is possible.

Begin by *checking the log book and the whiteboard for any special instructions* before proceeding with the following. Note that there are separate log books for Maxc1 and Maxc2. The last few log book entries may describe a crash like the one that has just occurred. This may suggest a restart procedure for you to follow.

You should *append a log entry to the logbook* with your name, the date and time, a statement that Maxc crashed, and any other information that you discover while following the procedures below. Recent error typeouts (e.g., memory parity error messages) should be cut out and taped in the logbook or copied from the display into the log if relevant.

*Maxc1*: Look carefully at the Infoton terminal (which has a sticker saying "Maxc1 Nova" pasted to it) and at the console teletype. The Infoton will show the "flashing-register" display and error messages from NVIO. Look at the logging terminal, which is a TI terminal off to the side of the Maxc room near the door.

*Maxc2*: Look carefully at the upper and lower windows on the Maxc2 Alto. The upper window will show the "flashing-register" display and error messages from AltIO; the lower window serves as a console teletype. Look at the Maxc2 logging terminal, a Diablo printer located behind the Maxc2 Alto console.

*Maxc1 Nova or Maxc2 Alto failures* are generally manifested by an NVIO punt (*Maxc1*) or Alto Swat call (*Maxc2*) or by the Nova or Alto hanging someplace. If it hangs, then the flashing-register display will not be updating TODCLK and none of the other registers in the flashing-register display will be being updated either.

The console teletype will show error messages from Tenex. The logging terminal will have numerous messages typed out by Tenex before it crashed. The "flashing register" display shows the names and contents of several Tenex core locations that are frequently updated during normal operation. For example, TODCLK contains the time-of-day, which is updated by NVIO/AltIO; this register should be getting updated regularly by NVIO/AltIO, if NVIO/AltIO hasn't crashed. The Tenex scheduler will be frequently updating the NBRUN-NBPROC word if Tenex is running normally.

Information about the crash may be apparent to you when you read the print out on these. The logging terminal may have BUGCHK messages from Tenex; frequently the BUGCHK messages are irrelevant to the crash, but sometimes they are interesting. The log is normally filled with messages about network failures of various types and recoverable disk errors such as:

**PUPSRV date time FTP: Server timed out ...
***IMPBUG 8599 Header ...
*DSKERR: ...

These messages are usually irrelevant to the crash, so don't be overly concerned about them. However, if the crash is caused by a network jamup of some kind, the BUGCHK messages on the logging terminal may give a clue to the nature of the problem (for Ethernet or ARPA network failures).

When system personnel are not present, Tenex is generally left in a mode in which it attempts recovery from Tenex-detected errors. Thus most crashes handled by non-system people will be of a more obscure (relative to Tenex) nature. The following paragraphs describe some of the more common crashes and suggested recovery procedures.

Your first objective in dealing with a crash is to determine what kind of failure caused the crash. To do this you will look at various symptoms and try to classify the failure. The following are plausible reasons for crashes:

a. *Ethernet problems* occasionally cause Tenex to be inaccessible, even though Tenex has not crashed. In this case, there will be no BUGHLT message from Tenex, no "Microcode halt" message from NVIO/AltIO, and the flashing registers will be updating normally. This situation is normally accompanied by numerous network-related BUGCHK messages on the logging terminal. You can determine whether or not this has occurred by typing control-C on the console teletype (*Maxc1*) or to the lower window of the Alto (*Maxc2*); if Tenex responds to control-C with a login message, then you know that it is still alive, and you should find someone to fix the Ethernet. (On Maxc2, if the cursor is not flashing in the lower window, you have to type the bottom unmarked key at the right of the Alto keyboard before typing control-C.).

b. *Software or firmware bugs* have been rare, usually related to the ARPA, Ether, and MCA networks, and these bugs generally manifest themselves only when the network hardware is malfunctioning in some way. However, a Tenex BUGHLT or microprocessor hanging condition might be caused by a software/firmware bug.

c. *Disk drive or disk controller failures* are frequent causes of crashes. These might be manifested by one of the disk drives going into select-lock (a red light on the disk control panel turns on when this occurs.). Building power glitches might also cause this. A disk drive/controller failure will normally show the symptom of a BUGHLT message on the console teletype (*Maxc1*) or lower window of the Alto (*Maxc2*). Disk failures are discussed below.

d. *Main memory failures* are generally manifested by a failure message from NVIO/AltIO. Garden variety uncorrectable double errors may result in the Tenex parity error sweep being invoked for a crash autorestart, and this may in turn be followed by a microprocessor halt as discussed below. Memory cabinet power supplies sometimes turn off due to shorts or other hardware failures, manifested by one of the four power supply lights on a memory cabinet turning off. An assortment of symptoms for various memory failures are discussed below.

e. *Microprocessor failures* are generally manifested by microprocessor halts, by peculiar Tenex BUGHLT's, or by the microprocessor hanging. If the microprocessor hangs, the TODCLK item in the flashing-register display will generally be updating normally (TODCLK is updated by NVIO/AltIO, not Tenex), but other flashing-register display items will be unchanging. This could also occur when the core image of Tenex is smashed in some strange way.

The various symptoms which imply one or another of these kinds of failures are discussed below.

A. **BUGHLT or BUGCHK:** A message of the form

```
BUGHLT at 73550
$8B>>BUGADR  BUGHLT/ CAI UUOH+4
```

types out on the Maxc console. This occurs when the monitor is in debug mode, which should never be the case unless system maintainers are present. However, if you cannot find one, set the DBUGSW and DCHKSW cells to zero and proceed from the breakpoint:

```
DBUGSW/ 1   0<lf>
DCHKSW/ 1   0<cr>
<esc>P
```

Tenex will recover from the error if possible, else restart automatically with no further intervention required

Sometimes Tenex will hang in the autorestart code which follows a "BUGHLT at nnnnnn" message. This may be indicated by TODCLK in the Tenex register display updating normally but nothing else happening. In this case, lookup the message associated with the BUGHLT in the BUGSTRINGS.MAXC1/2 listing on the table. This frequently is caused by a disk or microprocessor hardware problem. Unless something more creative occurs to you when you read the BUGHLT message, try the "Last Resort" procedure in paragraph J below.

B. **"Trouble with System Pack nnn"** prints out on the Maxc console, followed by "Type M to move pack, R to resume". This is caused by a disk unit going offline or failing in some equally catastrophic way. If the source of the problem is obvious (e.g. someone switched the unit off accidentally), rectify the problem, wait for the unit to be online (green light lit), and type "R" on the Maxc console. In other cases, it is usually better to move the pack to a free drive (if there is one--frequently the only free drive has a Bsys backup pack mounted on it, which you may remove). After waiting for the new drive to be online, type "M" followed by the letter corresponding to the drive you have moved the pack to (A through H). Tenex should now resume automatically.

Sometimes disk units have gone into select lock without any apparent reason (e.g., after a building power glitch). This can be cured sometimes by powering down the unit, letting it stop, then powering up again. *After powering down the front panel switches and waiting for the disk unit to stop*, you may have to turn off the AC power switch in back in order to clear select lock. If the disk unit *does not stop* when you power down the front switch, do *not* turn off the AC power in back because the heads may not have retracted, and you will destroy the disk pack by powering down. We have had several failures like this.

If moving the pack doesn't succeed in restarting Tenex, or if another crash occurs later, you will have to restart by booting Micro-Exec. Then you will have to tell Micro-Exec what the new disk configuration is. This is done by using "Print.Disk.Configuration" and "Set.Disk.Configuration" commands as discussed in the Micro-Exec section of this document.

If by unfortunate chance the drive that fails is the *first* one in the old configuration, then Micro-Exec will be in the first save area on that drive, and you will have to boot it by typing nB to NVIO (Maxc1) or AltIO (Maxc2) as discussed in the AltIO and NVIO sections. This is a little different from the normal boot procedure which defaults the drive for booting to drive A.

C. **"Micro Breakpoint"** prints out on the Nova console (Infoton) on Maxc1, or in the AltIO command window on Maxc2. (On Maxc1, this and similar NVIO messages will usually appear *above* the lowest line of text and two lines of numbers usually displayed by NVIO.) This message means that the microprocessor hit a breakpoint, which is usually caused either by Tenex executing a HALT instruction or by the microcode detecting some serious internal inconsistency.

The only known HALT instructions in Tenex are associated with catastrophic disk errors, and a message such as

> IRREC. READ ERROR IN DIRECTORY--BEWARE OF DISK WRITE FAILURE
> TROUBLE WITH DISK PACK 000211

is typed out on the Maxc console. Errors of this nature should be handled only by knowledgable system people, since the Tenex file system may be endangered.

A "Micro Breakpoint" not accompanied by a printed message on the Maxc console is usually due to a microprogram-detected inconsistency. Perform the following procedure:

1. Enter Midas by typing the following on the Nova console or Alto keyboard:

   *Maxc2:* Strike the middle unmarked key.
   #3301P (to "un-protect" NVIO/AltIO)
   :M...OK. (to enter Midas)

2. Write down the contents of the following registers displayed by Midas:

   NPC IMA P Q STK 0 PC PISTAT F INSTR

3. *Maxc2 only.* Execute the "Compare" command in the command menu (you must confirm it with Return). If this prints "No errors" or "1 errors on Midas.Errors" then the microcode is ok, so continue at step 6 below. Otherwise, exit to the Alto Executive with "Exit", issue the command "Type Midas.Errors", and write down anything interesting. (The microcode legitimately clobbers SM location IODEND, so if this is the only error in Midas.Errors then nothing is wrong.) If there are any real errors, most likely a bipolar memory chip has failed, and attempts to restart the system will probably be unsuccessful until the chip is replaced. Notify a hardware maintainer.

4. Type the commands "21;G" (which should end up within a few seconds at IMA=30), followed by "25;G", which checks the correctness of the microcode. If IMA=30, the microcode is ok and you should go on to step 6. If IMA=20, the microcode is incorrect. Write down the contents of LM 10. If you are ambitious, consult Section 13 for information on interpreting Checker failures. Run appropriate microprocessor diagnostics if you are familiar with them.

5. *Maxc1*: Type control-A to return control to DOS. Then reload the microcode via the command:

   MIDAS TENLOAD <cr>

   This takes a while (about 2 minutes). Wait until all messages at the bottom of the screen disappear.

   *Maxc2:* Successively select the menu items "Run-Program" and "Tenload" using the left mouse button.

6. Attempt to "soft-restart" Tenex as follows:[1]

   *Maxc1:*
   !NVIO.SV/H<cr>
   NVIO
   :140G...OK.

   *Maxc2:*
   Select menu items "AltIO", "Dont-Go", "Do-It". Then type:
   :140Go [confirm] .

   If this is successful, Tenex will within a minute or so broadcast the message "Maxc resumed from service interruption" to all terminals. If not, follow the instructions in "Last Resort" (paragraph J below).

D. **"Bipolar Memory Parity Error"** (*Maxc2 only*). Handle as in case C, except that interpreting the Checker failure is an especially desirable thing to do. Perform a "LMPEscan" before doing step 3, and write down any errors reported. If bipolar memory parity errors keep occurring, notify a hardware maintainer, since it is necessary to change a bipolar memory chip.

E. **"Fatal Memory Error, Maxc Stopped"** on Nova/Alto console. This indicates that the memory is very sick, and a hardware maintainer should be notified.

"Main memory error: DE q" (*Maxc2 only*) where q = J, K, L, or M indicates a main memory storage problem in the indicated memory quadrant.

"Maxc halted with memory bus parity error" (*Maxc2 only*) indicates a problem in the logic for generating or sending parity from the memory to the processor or in receiving parity by the processor, or in transmitting one of the data bits between the memory and the processor. It is normal for this to occur in conjunction with a "Main memory error."

"DIP in Q" (where Q = J, K, L, or M) means that the parity of the data on the bus from the port to the processor was incorrect. This will happen in conjunction with a "DE in Q" and isn't significant in this case. In other cases it indicates a hardware problem in the port or in the transmission path from the port into the processor.

*You should check the power supply lights before embarking on any other action.* Enter the machine room and locate the bank of logic racks for the Maxc machine which has crashed. You will see the Maxc1 Nova or Maxc2 Alto (there are signs on them). To its right will be the cabinet containing the processor and port (labelled "Maxc1 processor" or "Maxc2 processor"). The two power lights at the bottom of the cabinet should be lit.

To the right of the processor are memory cabinets (presently 3 cabinets for each system). The right-most four lights at the bottom of each memory cabinet should be lit. (The two lights to the left of these are insignificant).

If any of the power lights is not lit there are three possibilities: the light has burned out; some electrical short has legitimately invoked the safety circuits and shut down the supply; or a glitch has invoked the safety circuits, but the hardware is ok (frequent source of failures). Because there may be an electrical failure, you should notify a maintainer if possible. However, if you can't find one, hope that nothing fatal has occurred and proceed as follows:

---

[1]Note that it is generally possible to "soft-restart" Tenex even after running microprocessor diagnostics such as DGBASIS andDGIML (but not DGM or DGMR, which are memory diagnostics).

First, put the system disk drives in Read-Only mode; it is necessary to do this before power down until after power up. Then, power down processor, port, and memory, as discussed in the Power Down section. This is done by running a program. Do not turn off any hardware switches. Then power up the processor, port, and memory as discussed in the Power Up section starting at step F.

*Note:* You have to power down Maxc before powering up again.

If this succeeds in getting the power supplies on again, try the cold start procedure for restarting Tenex as discussed in Paragraph J. If it does not succeed, then the hardware is broken and has to be fixed.

If the lights are all on, and if you can't locate a hardware maintainer, you should restart Tenex from scratch. If the failure is a double error caused by failure of storage components, then the restart procedure will zone out the bad storage region so that Tenex will not use that area and the failure will not reoccur. If the failure is more serious, such that the memory is unusable, then the hard restart will fail and the hardware will have to be fixed. Paragraph J below discusses the hard restart procedure.

F. **"NVIO Punt"** on Nova console (*Maxc1 only*). If this is an immediate punt after running DGM or DGMR, do "POWER ON" and try again. Otherwise, this indicates a serious inconsistency detected by NVIO. Crash data should be saved and the crash recovered as follows (type on the Nova console).

```
#3301P                  ("un-protect" NVIO)
:D...OK.                (enter Nova debugger)
XPUNT/ junk :sssss+n = xxxxxx <lf>
PUNT0/ junk :sssss+n = xxxxxx <lf>
PUNT1/ junk :sssss+n = xxxxxx <lf>
PUNT2/ junk :sssss+n = xxxxxx <cr>
<esc>P                  (Resume NVIO)
:R...OK.                (Resume Maxc)
```

If Tenex does not resume after this procedure, try a "soft restart" by typing:

```
:140G...OK.
```

on the Nova console.

Write down in the logbook the data typed out by the debugger in response to the ":" and "=" characters you typed in.

G. **Nova crash** (*Maxc1 only*). If NVIO has stopped updating the bottom row of numbers on the Infoton screen, it is most likely that the Nova has crashed. Go into the machine room and record the Nova's state, as follows:

1) If the Nova is still running ("Run" lit), press "Stop" followed by "Continue" a few times, recording the state of the "Address" lights after each "Stop". Leave the Nova stopped.

2) Record the state of the "Address" and "Data" lights.

3) Make sure the switches are set to 100040. Then press "Reset" followed by "Start". The Infoton should print out a row of numbers (if not, go to step 4). Write these down in

the log. Then type <esc>P. After a few seconds, the message:

BREAK
R

should print out. Then type:

SAVE CRASH <cr>

which saves the crashed core image for later examination.

4) If the procedure described in step 3 failed, press "Reset" followed by "Program load" on the Nova front panel. Then, in either case, attempt the "soft-restart" procedure described above (paragraph C, steps 5 and 6).

We have had periods when the Nova disk gets smashed occasionally. A crash in which the disk is smashed might manifest as being unable to boot the machine. If a disk crash is suspected,

1) Take down Portola (or some other two-disk Nova);

2) Put good disk in dp0, bad disk in dp1;

3) Boot the machine and then turn off write protect by pushing the red buttons on dp0 and dp1.

4) Run DKUTIL and type G↑c to start.

This will copy the good disk onto the bad disk.

**Alto crash** (*Maxc2 only*). If the Alto has fallen into Swat (the message "Swat" followed by a number and a date appears at the top of the screen), record in the log book the information below the lowest line of squiggles. Then press the boot button on the back of the Alto keyboard. Then issue the commands:

AltIO/H <cr>
140Go [confirm] .

This should result in a Tenex "soft restart", as in Paragraph C, step 6.

H. **"Disk needs fixing" message from CHECKDSK.** When Tenex autorestarts following a BUGHLT, it first runs the BSYS verify and CHECKDSK programs to determine whether or not the file system has been damaged by the crash. If either of these programs detects a problem, it will abort the autorestart. Fixing these problems is hazardous and should ordinarily be attempted only by a system maintainer. The procedures for recovering from CHECKDSK failures are discussed in a later section.

I. **No response from Tenex**; i.e., no error messages have typed out and none of the above alternatives seems to apply, but nothing happens when you type control-C on the Maxc console. This type of crash is particularly hard to diagnose unless sufficient information is recorded.

First, note the numbers on the last line of the Nova console (*Maxc1*) or at the top of the Alto screen (*Maxc2*), and note whether any of them are changing over time.

Second, enter Midas by typing the following on the Nova console or Alto keyboard:

*Maxc2:* Strike the middle unmarked key.
#3301P (to "un-protect" NVIO/AltIO)
:M...OK. (to enter Midas)

(If the message "Unclean Micro Stop" prints out, note this as well.)

Now continue by carrying out the instructions given earlier beginning at paragraph C, step 2.

J. **Last resort.** It may happen that a crash does not fall into one of the above categories or that the restart procedure fails. In this case, the following procedure will *always* succeed if all the hardware is working:

1. *Maxc1:* On the Nova front panel in the machine room, make sure the address switches are set to 100040. Then press "Reset" followed by "Program Load". On the Nova console (Infoton), you should see:

   DOS REV 04
   R

   Then type "POWER ON <cr>"

   *Maxc2:* Boot the Alto.

2. Type the command:

   MIDAS TENGO <cr>

3. Wait about 2 minutes (*Maxc1*) or 30 seconds (*Maxc2*) while the microcode loads and NVIO/AltIO and Micro-Exec are started. Micro-Exec will execute an automatic "Go" command, after which you should enter date and time if Tenex requests it. Generally, you can ignore bad-chip messages printed out during memory testing--the regions of storage affected by bad chips are mapped out by Tenex. Save the printout in the log book, however.

If this doesn't work, try to find any one of the people listed below at PARC. If none of them is around and the hour is between 9 AM and midnight, call one of the system maintainers. If between midnight and 9 AM, don't bother, but leave a message on the telephone recording saying that the machine will be down until morning. Instructions for recording messages are posted in the back room.

**People to notify:** (use phone list on wall beside phone)

Software and general system operation

| | |
|---|---|
| Taft | General, Tenex, NVIO, and AltIO |
| Fiala | General, microcode, Midas, Tenex |
| Boggs | General |
| Geschke | General |

Hardware

| | |
|---|---|
| Fiala | Microprocessor |
| Lampson | Microprocessor |
| Overton | Memories and Disks |

| | |
|---|---|
| McCreight | Disks |
| Yeary | Alto |
| Quaterman | Nova |
| Mann | Nova |
| Winfield | Nova |
| Thacker | Almost anything |
| Taft | When any of the above aren't available |

## 3. POWER UP

A. First look over the system and make sure it is put together. Ignore scope probes dangling from pins, but at any other appearance of inoperativeness, give up or get some knowledgeable person to help.

B. Make sure the proper disk packs are mounted. The labels are on the top of the pack. 3-by-5 cards with the currently mounted pack numbers are supposed to be posted on each drive. Other packs are in a yellow cabinet in the machine room and should also have cards identifying them.

C. Turn on the disk units. Two switches need to be turned on for each unit. One of them is in back and inside the cabinet and is labelled "AC power on". It should be turned on first. (Normally it is left on.) Then turn on the obvious switch on the front panel. Two minutes will elapse before the green light goes on. The read-only switches on the control panel should also be off.[1]

D. *Maxc1:* Turn on Nova power (the key on the front panel). Load the disk cartridge labelled "Maxc Nova" into the disk drive over the Nova and flip the switch to "Run" and press the red "protect" light. Turn on both magtape controllers and all 3 tape drives by means of their front-panel switches. Turn on the DLS by pressing the circuit breaker on the back of the DLS power supply.

*Maxc2:* Turn on Alto power by pressing in the circuit breaker on the front panel. Load the disk cartridge labelled "Maxc2" into the disk drive under the Alto and flip the switch to "Run". Make sure the Alto monitor and the Diablo printer are turned on.

E. Turn on the microprocessor fan, two microprocessor power supplies, and all memory power supply switches in front of Maxc. (Normally these are left switched on. The power supplies are turned on and off under program control.)

F. Assuming the Nova or Alto disk is loaded with the correct contents, DC power to the Maxc processor and memories may be turned on by this procedure. (The next section describes loading the Nova disk from tape.)

*Maxc1:* Set the Nova's front panel switches to 100040, then press "Reset" followed by "Program Load". On the Nova terminal (Infoton), issue the command:

POWER ON <cr>

This turns on the power supplies and configures the memory; the operation takes about 15 seconds. The program will complain about any supplies it is unable to turn on. *Note:* Maxc1 has a hardware problem in the power on/off control, and it is possible for the memory system to power up in an inoperable state--see the section on Tenex crashes for symptoms and cures for this problem.

*Maxc2:* Boot the Alto and run the Midas subsystem. With the left mouse button, successively select the menu items "Power On", "Both", and "Do-It".

---

[1]Note that if a unit is *selected*, the read-only restriction will remain in force until it is deselected. This is indicated by the read-only light on the panel. Don't worry if the read-only light remains on, since the unit will be deselected by firmware later on and the read-only light will go off at that time. However, be sure the read-only switch is in the off position.

## 4. POWER DOWN

You should *not* power down the processor or the memories by pulling plugs or throwing switches since this sudden shut down can damage components. Only resort to this drastic action in an emergency (e.g., fire or when the Nova/Alto is broken so that the program for powering down cannot be executed). If you *do* have to turn power off manually, try to turn all four switches for each particular memory cabinet off as sumultaneously as possible.

On *Maxc2*, the front panel switches for the disk drives should be turned off before powering down the microprocessor. On *Maxc1*, this is desirable, but leaving the disks on hasn't been observed to cause problems.

After powering down the disks, then shut off the microprocessor and memories as follows:

On *Maxc1*, proper shut down is accomplished by issuing the Nova DOS command:

POWER OFF <cr>

*NOTE: At present the processor power on Maxc1 cannot be turned ON or OFF under program control. Throw the main breaker on the front panel.*

On *Maxc2,* power is turned off by running Midas and selecting the "Power-Off" and "Both" menu items. Temporarily (as of 10/22/79), Midas cannot turn off the Port power supply on Maxc2. To turn it on and off you must throw the front-panel switch labelled "Port on-off."

When you power off, the two red lights on the processor and the four red lights on each memory will shut off. The fans will continue to run and should be left on for at least 15 minutes. You may then turn them off by pulling out their plugs in the backs of the cabinets, or by turning off the appropriate circuit breakers in the back room.

It is possible to turn the processor and memories on and off independently. If only the processor needs to be turned off, it is best to leave the memories on so as to avoid shortening the life of the storage chips.

On *Maxc1*, type "POWER OFF/P" to turn off only the processor, and "POWER ON/P" to turn it on. Similarly, "POWER OFF/M" and "POWER ON/M" turn off and on only the memories. On *Maxc2,* this choice may be made by means of the subsidiary menus for the "Power-On" and "Power-Off" commands.

To totally turn off disks, turn off the front panel switches first. Do not turn off AC power (in back) until the pack has stopped turning, else the drive may stop with the heads still extended! Unless you are going to work on the disk drive, powering off the AC power in back is not required.

If the Nova/Alto won't run, then you will have to power Maxc down manually. It is very important to turn off all Maxc power supplies before turning off the Alto power. To do this, first turn off the Maxc disk drives via the switches on the front panel. Then turn off the four power supply switches on each memory cabinet. Turn the four switches on each cabinet off *as nearly simultaneously as you can* to avoid possible problems in the power safety circuitry. *After* you have powered down the memory cabinets, turn off the processor and port power supplies.

Finally, turn off the Alto disk, and when it stops the Alto processor. If the processor is going to be powered off for a long time, you should also power down the Alto display.

For most repairs to the Maxc processor, it is not necessary to power down the Alto. However, if you are going to disturb the PMAINT, KMEMI, or PMEMI cards in the processor, it is imperative that you power down the Alto (after powering down Maxc). It should be ok to leave the Alto running when other processor cards are pulled.

## 5. LOADING THE PDP-10 EMULATOR

Assuming system power has been turned on and the Nova or Alto is at command level (Nova DOS or the Alto Executive), one may load the microcode and start Tenex by means of the single command:

MIDAS TENGO<cr>

Midas will run through the TENGO command file which loads the microcode, displays some junk on the screen, and starts up Tenex. This takes several minutes on Maxc1 and about a minute on Maxc2.

After loading the microprocessor, TENGO resets the machine (21;G) and checks the microcode (25;G). The microprocessor may crash (IMA=20) if there were any failures detected during loading. LM 10 will contain the address of the incorrect word (if only one word was clobbered).[1] Note the values of LM 10 and IMA in the log and call system maintainers if the microprocessor crashes. If you cannot get the system maintainers, get out of Midas (by typing control-A on Maxc1 or selecting "Exit" with the mouse on Maxc2) and repeat MIDAS TENGO, since hardware flakiness has been known to cause loads to fail spuriously and you may be lucky on the second try.

If the load succeeds, NVIO or AltIO is started, which in turn boots in Micro-Exec from one of the Maxc disks. The following message is printed out on the Maxc console:

Micro-Exec, Ver <version identifying string>
*

Micro-Exec immediately executes an automatic "Go" command, which causes Tenex to be loaded from disk and started. Further procedures are described in the next section.

If it is desired to load the microcode without starting NVIO/AltIO or Micro-Exec, use the TENLOAD command file instead of TENGO; i.e. type:

MIDAS TENLOAD <cr>

TENLOAD is identical to TENGO except that it does not start NVIO/AltIO, Micro-Exec, or Tenex, but rather leaves control in Midas.

*Maxc1:* Then, to start NVIO and boot in Micro-Exec (without starting Tenex), type:

!NVIO.SV/B <cr>

*Maxc2:* To start AltIO and boot in Micro-Exec (without starting Tenex), select "AltIO", "Boot-MExec", and "Do-It" with the mouse.

The MEXECGO command file combines the effect of TENLOAD followed by starting NVIO or AltIO and booting in Micro-Exec.

*Note:* The above procedure only works when Micro-Exec is on save area 1 of disk unit A. This is normally the case. However, during periods when disk controllers are being checked out or when there has been hardware flakiness with disk controllers, MEXEC may not be on unit A. In this case it is probably on another disk drive. You should follow the procedure discussed in "Calling NVIO" (Maxc1) or "Calling AltIO" (Maxc2) in order to boot Micro-Exec from another drive.

---

[1] Checker failure interpretation is discussed in more detail in section 13.

## 6. STARTING TENEX

The current Micro-Exec normally has been setup to have the current disk configuration. If you doubt this, type:

Print.Disk.Configuration <cr> (P D C)

and it will print out the current disk configuration. Compare this to the numbers on the mounted disk packs. If you are confident, type:

GO<cr>

which will first run a brief (~30 second) memory test, then boot in Tenex from the disk and start it running.

If the disk configuration printed does not agree with the actual then type:

Set.Disk.Configuration <cr> (S D C)

which asks for parameters via the same format used in print.disk.configuration.

After setting the correct disk configuration you must write MicroExec to area 1 by typing:

Write.Micro.Exec.To.Area <Area> 1<cr> (W M E T A.1)

Now you may type:

GO<cr>

Tenex may ask for the date and time. (The prompt specifies the format.) Please be careful to enter this correctly. If Tenex asks you to reconfirm your typein, you probably blew it and you should hit "Del" and try again. (But if you are really sure you typed the correct date and time, confirm with carriage return).

Tenex then runs the Bsys and Checkdsk programs to verify the consistency of the file system. This takes about 15 minutes (the time is proportional to the number of files in the entire system), at the end of which is broadcast the message "Tenex in operation" if all is well. If any uncorrectable errors have been detected, the message "Tenex not available: Disk needs fixing" will be broadcast. If this occurs, you should attempt to notify system personnel. As a last resort, consult Section 18 for information on fixing errors of this sort.

If Tenex was last taken down in an orderly fashion (as opposed to crashing), it is not really necessary to run Bsys and Checkdsk (though it will never hurt to do so). The running of these programs may be bypassed (saving 15 minutes) by the following procedure:

1. Load the microcode using the MEXECGO command file, as described at the end of the previous section. This causes Micro-Exec to be started on Maxc.

2. Manually run the Micro-Exec memory test (so that Tenex will be informed of solidly failing regions of memory that it shouldn't use):

    *Test.Memory.Fast <cr>  (T M F)

3. Read in Tenex without starting it:

    *Read.Tenex.From.Area 0 <cr>  (R T F A 0)

4. Enter Exec DDT, turn on the "Checkdsk bypass" switch, and start Tenex:

```
*EDDT  <cr>
EDDT
CHKBYP/   0       1  <cr>
SYSGO1<esc>G
```

## 7. STOPPING TENEX

In order to give users adequate notice and to protect files on disk, Tenex should ordinarily be stopped in the following manner.

As long as possible before the scheduled downtime (preferably 24 hours or more), notify users by (optionally) putting a notice in the login message, setting the system downtime cell, and recording a telephone message. Make sure you do these things on the correct Maxc system!

A. To put a notice in the login message,[1] login as yourself and:

```
@SNDMSG <cr>
To: SYSTEM <cr>
cc: <cr>
Subject: Scheduled downtime <cr>
Message:
(An appropriate message giving date, time, and reason)
↑Z
Q,S,?,carriage-return: <cr>
SYSTEM -- ok
@
```

---

[1] Do this only if the downtime is of unusual nature or duration. You must be a member of the "System" group to send messages to SYSTEM.

B. To set the system downtime cell, you must be a wheel or a maintenance person:

```
@ENABLE <cr>     (only if you are a wheel)
!HALT, <cr>      (do not omit the comma)
[Superpassword:] GUESS <cr>
!!AT mm/dd/yy hh:mm <cr>                          (date&time system going down)
!!UNTIL mm/dd/yy hh:mm <cr>                         (date&time coming back up)
!!DUE (TO REASON) reason (type ? for list) <cr>
!!<cr>
!
```

C. To record a telephone message, go to the recording telephone (next to the Imp in the back room) and follow the instructions posted there. Make sure you specify Maxc1 or Maxc2 in the message.

The system will automatically start notifying users of the impending downtime beginning one hour before it is to occur. When the zero hour arrives, all jobs will be forcibly logged out except any job logged in on the Maxc controlling terminal, new logins will be prevented, and "Shutdown Complete" will type out on the Maxc console. If there are now no jobs logged in, Tenex will shortly hit a BUGCHK (EDDT breakpoint) at SWHLT1, at which point Tenex is properly halted.

If there is a job logged in on the Maxc controlling terminal, it will be necessary to halt Tenex manually. To do this (for which you must be a *wheel*):

```
@ENABLE <cr>
!QUIT <cr>
.Halt Tenex.
```

Wait for the EDDT breakpoint at SWHLT1. A message such as "$8B>>CHKADR BUGCHK/ SWHLT1" should be printed out. At this point it is safe to un-protect the NVIO/AltIO console ("3301P") and return control to Midas ("M").

## 8. NVIO AND ODT

*Maxc1 only.* NVIO is a Nova Input-Output package which does all the input-output for Maxc except for the disks. Specifically, it does input-output for the data line scanner, the IMP (ARPA Network), the MCA (Multiprocessor Communications Adaptor), the Ethernet, and the tape units. It also has an octal debugger called ODT (Octal Debugging Tool) which will allow the user to change memory locations in Maxc and the Nova, and perform other control functions.

### 8.1. Calling NVIO

NVIO can be called from Midas with the command:

6,NVIO;T

or:

!NVIO.SV/a/b ... <unitno><cr>

where /a/b etc. represents switches (of the usual Nova DOS type); and <unitno>, if it exists, is of the form 'Un' for n=0 to 7. The NVIO startup procedure is determined by the setting of these switches, the function of each of which is now defined -- in the order in which they are tested to avoid any ambiguity caused by multiple switch settings. Most of the switches have an analogous ODT command with the same (letter) name.

/R Reset the Maxc memory system. This is a fairly catastrophic operation and should be performed only if the memory is hung up.

/B Boot Micro-Exec from Save Area 1 of disk unit n (from the 'Un' argument), where 0=A, 1=B, ... 7=H. Unit A is used if no unit is specified.

/S Start Tenex, first booting Micro-Exec from disk as for /B and then executing an automatic "Go" command.

/E Bring in the Maxc Micro-Exec from tape unit number n (from Un) or from unit 0 if no unit is specified.

/P Enter Protect mode in ODT. This means that ODT will allow only those commands that examine (but do not write into) memory and that do not change machine state.

/D Enter the Nova Debugger. ODT Control may be resumed by typing $P.

/H Normally NVIO now sends a "proceed" to Maxc which was probably halted before NVIO was called. This auto-proceed can be suppressed by using the /H switch.

/W NVIO will Wait for an IORESET before proceeding.

Normally, NVIO is initially started up with the /B or /S switch set, in order to load Micro-Exec and possibly start Tenex. When it is desired to start NVIO without resuming Maxc (e.g. to examine Maxc main memory or start at an alternate address), the /H switch should be used.

### 8.2. ODT Commands

When NVIO is functioning, ODT (Octal Debugging Tool) is its highest priority process (only interrupts take precedence). It has two heralds, a number sign (#) which indicates that only commands that examine memory are legal, and a colon (:) which indicates all commands are legal.

If Tenex is running and no ODT commands have been typed for ten seconds, NVIO will continuously display and update the addresses and contents of five Maxc main memory variables. If you start typing on the Infoton, NVIO will stop doing this and will print the appropriate herald followed by your typein on the next line.

ODT commands consist of a single letter or other special character, optionally preceded by a numeric argument. The argument is an octal number of 16 or 40 bits (depending on the command) or an expression made up of such numbers and the operators "+" and "-". "." stands for the current (most recently displayed) memory address.

Many NVIO commands request confirmation with "...OK". The confirming character is a period. The NVIO commands marked with a superscript "1" are valid and have the same meaning in AltIO.

n] Change the current machine to the Nova; change the current memory location to n; and print out the contents of that location.

n[ Change the current machine to Maxc; change the current memory location to n; and print out the contents of that location in the format:

aa bbbbbb cccccc

where:

aa (0 to 17) is the last 4 bits (i.e., the "tag" bits) of the 40-bit Maxc memory word, it is not printed out if zero;

and

bbbbbb and cccccc are the left and right half 18 bits of the PDP-10 (Maxc) word.

n/[1] Change the current memory location to n; and print out the contents of that location.

/ Print out the current location.

n<cr>[1] Put n into the current location. If the current machine is the Nova, a 16-bit number is permissible. If the current machine is Maxc, a 40-bit number is permissible. The first 4 bits input are the last 4 bits of the memory. Thus one can think of Maxc as a 36-bit machine (like the PDP-10), right adjusted in a 40-bit field, rather than a 36-bit machine left adjusted in a 40-bit word as is actually the case.

This command is only legal after a command which prints out a Maxc or Nova location.

<lf>[1] (lf means linefeed). Change the current location to the current location plus 1, and print out the (new) current location.

n<lf>[1] Equivalent to n, <carriage return>, <linefeed>.

↑[1] Change the current location to the current location minus 1, and print out the (new) current location.

n↑[1] Equivalent to n, <carriage return>, <up arrow>.

<tab>[1] Change the current location to the location pointed to by the current location. Print out its contents.

B[1] Boot Micro-Exec from Save Area 1 of physical disk unit A.

nB[1] Boot Micro-Exec from Save Area 1 of physical disk unit n (0=A, 1=B, etc.) The parameter n is remembered for subsequent "B" commands without arguments.

D Go to the Nova Debugger. $P will resume ODT.

E Startup the Maxc Micro-Exec, loading from Tape Unit 0.

nE Startup the Maxc Micro-Exec, loading from Tape Unit n.

nG[1] Go to Maxc location n. Precisely: stop the Maxc microprocessor (cleanly if possible), then tell it to begin executing PDP-10 instructions at Maxc main memory location n.

H[1] Halt Maxc (cleanly if possible).

I Initialize NVIO.

nI Execute initialization routine n:

- 0 Initialize all devices
- 1 Initialize DLS
- 2 Reset memory system

M[1] Halt Maxc (cleanly if possible) and return control to Midas.

nM[1] Go to Microprocessor location n. (If you use this command, you had better know what you are doing.)

P[1] Protect NVIO. The herald becomes a number sign (#); only those commands that examine bmemory (and other state varibles) are legal.

nP[1] If the number typed is the Pass(word) number: ODT becomes unprotected; the herald becomes a colon (:) and all commands are legal. Currently the pass(word) number is 3301; this should be easy to remember as it is the smallest number which is the sum of two different cubes in two different ways. ($14^3 + 1^3 = 11^3 + 12^3$; octal of course.)

Q Print out the current state of Maxc (running, stopped, or at a micro breakpoint).

R[1] Resume Maxc (the microprocessor is told to proceed). If a reset was previously done ("21M" to ODT or "21;G" to Midas), PDP-10 interpretation will begin at the starting address pointed to by main memory location 7; otherwise, Maxc will resume from where it was halted.

nR If n is zero, disable Maxc main memory error recovery; NVIO will halt instantly on any fatal Maxc error. If n is nonzero, reenable error recovery.

S[1] Startup Tenex, booting Micro-Exec from disk unit A.

nS[1] Startup Tenex, booting Micro-Exec from disk unit n (0=A, 1=B, etc.)

nT Begin Testing output on DLS line n.

nU Unhang Nova device number n by simulating a completion interrupt. If you use this command, you had better know what you are doing.

nZ Stop testing DLS line n.

### 8.3. Nova Locations of Interest

50: Version number. This number is incremented by 100 for each assembly of NVIO.

51: Normally this location has a -1. If it has the number n in it then NVIO will PUNT if it attempts to read or write into Maxc physical page n.

52: The pass(word) number. Currently this is set to 3301 as explained under the ODT command P.

53: Date of most recent NVIO assembly, in the form MMDDYY, where MM and DD are octal but YY is decimal (at least until 1978).

55: Save area to boot Micro-Exec from (usually 1).

56: Default disk unit to boot Micro-Exec from (initially 0, but changed by arguments to B and S commands).

57: Correction for known clock error, in seconds per day. This number is positive if the clock is known to be fast, negative if slow.

The following symbolic locations are of interest:

RTN:
M: location of code to return to Midas. The Nova debugger commands RTN$R or M$R are equivalent to the ODT command M.

BREAK: location to goto to make a save file of the current version of NVIO. After BREAK$R is executed, control will go to the Nova Exec and the command:

```
SAVE name
```

should be executed to save NVIO on the specified file. This is normally done after NVIO is patched by the following sequence:

```
DEB NVIO
make patches
```

```
BREAK$R
R
SAVE NVIO
```

If control goes to a debugger breakpoint after the BREAK$R is executed, this means that there is no disk space available. Exit via RTN$R; make disk space; and repeat patches. (Sorry!)

### 8.4. NVIO Punts

For each PUNT, the location at which the PUNT occurred is put into Nova register 3, and information particular to each PUNT is sometimes stored in register 0. Maxc is then halted (cleanly if possible) and "NVIO Punt" is printed on the Nova console.

Accumulators 0, 1, 2, and 3 are stored at PUNT0, PUNT1, PUNT2, and XPUNT respectively, and may be examined by going into the DOS debugger with the "D" command. After resuming NVIO with "$P", it is necessary to restart Tenex in one of the following (increasingly drastic) ways:

1) R...OK. (Simply tells Maxc to proceed).

2) :21M...OK. (Resets Maxc microprocessor).
   :140G...OK. (Attempts "soft" restart of Tenex).

3) S...OK. (Reloads and restarts Tenex).

More detailed information on handling NVIO punts and Nova crashes may be found in Section 2.

## 9. ALTIO

*Maxc2 only.* AltIO serves the same purpose running on the Maxc2 Alto as NVIO does on the Maxc1 Nova, though it has a much smaller complement of devices to service (just the Alto monitor, the Diablo printer and the Ethernet).

### 9.1. Calling AltIO

AltIO may be called either from Midas or directly from the Alto Executive. In Midas, selecting "AltIO" with the mouse causes a menu of AltIO subcommands to appear. Selecting "Do-It" causes AltIO to be started with no options selected; in this case, AltIO performs no initialization but simply resumes the microprocessor at its current location. If other subcommands are selected before "Do-It", they modify the initial actions of AltIO in various ways.

When AltIO is called directly from the Alto Executive, the same options may be specified by switches. The general form of the AltIO command line is:

AltIO/switches parameter/switch parameter/switch ...

The switches immediately following "AltIO" correspond to Midas "AltIO" menu items in the following way:

| Switch | Menu Item | Description |
|---|---|---|
| /B | Boot-MExec | Boot Micro-Exec into Maxc memory and start it. |
| /H | Dont-Go | Leave Maxc halted rather than resuming it. |
| /P | Protect | Enter "protected" mode in AltIO. This means that AltIO will allow only those commands that examine (but do not write into) memory and that do not change machine state. |
| /R | Reset-Memory | Reset the memory system. |
| /S | Start-Tenex | Start Tenex, first booting Micro-Exec from disk as for /B and then executing an automatic "Go" command. |

The optional additional parameters may be specified only in the Executive command line; there is no way to invoke them from Midas.

| | |
|---|---|
| filename/L | Load a PDP-10 "save"-format file into Maxc memory (default extension .SAV). Used for loading PDP-10 diagnostics when Micro-Exec won't work. |
| n/U | Use disk unit n rather than unit 0 in Boot-MExec and Start-Tenex commands, where 0 corresponds to unit A and 7 to H. |

### 9.2. AltIO Commands

When AltIO is running, the Alto monitor is divided by black lines into three windows. If Tenex is running, the top window continuously displays the addresses and contents of five interesting Maxc main memory variables. The middle window belongs to a command processor for controlling AltIO itself. The bottom window implements the Maxc controlling terminal used by Tenex and by stand-alone programs such as Micro-Exec and PDP-10 diagnostics.

Keyboard input is directed to whichever window currently contains the blinking cursor. The middle and bottom unmarked keys on the right side of the keyboard may be used to direct input to the middle and bottom windows respectively. Keyboard input is automatically switched to the bottom window when Maxc is started by commands such as "Boot" or "Go", and to the middle window when Maxc stops for any reason.

The AltIO command processor has two heralds, a number sign (#) which indicates that only commands that examine memory are legal, and a colon (:) which indicates all commands are legal.

AltIO commands consist of a single letter or other special character, optionally preceded by a numeric argument. The argument is an octal number of up to 40 bits, depending on the command. While typing an argument, one may use the editing characters control-A, control-W, and Delete to erase, respectively, a single character, a word, or the entire command.

Many AltIO commands request confirmation with "[confirm]". The confirming character is a Return or period. Most of AltIO's commands are intentionally the same as in NVIO; the ones for which this is true are marked with a superscript "1".

n/[1] Change the current Maxc memory location to n; and print out the contents of that location in the form:

aa bbbbbb cccccc

where:

aa (0 to 17) is the last 4 bits (i.e., the "tag" bits) of the 40-bit Maxc memory word, it is not printed out if zero;

and

bbbbbb and cccccc are the left and right half 18 bits of the PDP-10 (Maxc) word.

/ Print out the current location.

n<cr>[1] Put the 40-bit number n into the current location. The first 4 bits input are the last 4 bits of the memory. Thus one can think of Maxc as a 36-bit machine (like the PDP-10), right adjusted in a 40-bit field, rather than a 36-bit machine left adjusted in a 40-bit word as is actually the case.

This command is only legal after a command which prints out a Maxc memory location.

<lf>[1] (lf means linefeed). Change the current location to the current location plus 1, and print out the (new) current location.

n<lf>[1] Equivalent to n, <carriage return>, <linefeed>.

↑[1] Change the current location to the current location minus 1, and print out the (new) current location.

n↑[1] Equivalent to n, <carriage return>, <up arrow>.

<tab>[1] Change the current location to the location pointed to by the current location. Print out its contents.

B[1] Boot Micro-Exec from Save Area 1 of physical disk unit A.

nB[1] Boot Micro-Exec from Save Area 1 of physical disk unit n (0=A, 1=B, etc.) The parameter n is remembered for subsequent "B" commands without arguments.

D Toggle on or off a switch causing typeout on the Maxc controlling terminal (the bottom display window) to be copied to the Diablo printer. This switch is normally off, but turning it on is useful when debugging Tenex or doing memory maintenance.

E Print out the current state of Maxc's PDP-10 emulator. This includes the program counter and all the accumulators. This command may be issued only when Maxc is stopped.

nG[1] Go to Maxc location n. Precisely: stop the Maxc microprocessor (cleanly if possible), then tell it to begin executing PDP-10 instructions at Maxc main memory location n.

G Go to the starting address of the currently loaded Maxc program, which is pointed to by Maxc memory location 7.

H[1] Halt Maxc (cleanly if possible).

L Prompt for a filename (default extension .SAV), then interpret that file as a PDP-10 "save"-format (not "ssave") file and load it into Maxc memory. The program's starting address is stored in location 7 so that a "Go" command without arguments will start the program. The file is expected to be in the 36-bit binary format defined by FTP. Retrieving a PDP-10 "save"-format file using the FTP program will cause such a file to be created.

M[1] Halt Maxc (cleanly if possible) and return control to Midas. Midas is resumed in whatever state it was in when control was last transferred to AltIO from Midas. This is true even if AltIO has been exited and other Alto programs run in the meantime.

nM[1] Go to Microprocessor location n. (If you use this command, you had better know what you are doing.)

P[1] Protect NVIO. The herald becomes a number sign (#); only those commands that examine memory (and other state varibles) are legal.

nP[1] If the number typed is the Pass(word) number: AltIO becomes unprotected; the herald becomes a colon (:) and all commands are legal. Currently the pass(word) number is 3301; this should be easy to remember as it is the smallest number which is the sum of two different cubes in two different ways. ($14^3 + 1^3 = 11^3 + 12^3$; octal of course.)

Q Quit: return control to the Alto Executive, after first stopping Maxc if it is running.

R[1] Resume Maxc (the microprocessor is told to proceed). If a reset was previously done ("21M" to ODT or "21;G" to Midas), PDP-10 interpretation will begin at the starting address pointed to by main memory location 7; otherwise, Maxc will resume from where it was halted.

S[1] Startup Tenex, booting Micro-Exec from disk unit A.

nS[1] Startup Tenex, booting Micro-Exec from disk unit n (0 = A, 1 = B, etc.)

W Print out the current state of Maxc (running, stopped, or at a micro breakpoint).

Z Zap Maxc memory: reset and reconfigure the entire memory system. Useful when the memory has become hung up. This command may be issued only while Maxc is stopped.

# 10. MAXC1 MIDAS OPERATION

## 10.1. Conventions

The memories accessible to Midas are called:

| | |
|---|---|
| MAIN | system main memory, lower 64K words only |
| LM | left register bank, 32 registers |
| RM | right register bank, 32 registers |
| SM | scratch memory, 512 registers |
| DM | PDP-10 emulator dispatch memory, 512 registers |
| DM1[1] | Byte Lisp emulator dispatch memory, 512 registers |
| DM2[1] | unused dispatch memory, 512 registers |
| IM | instruction memory, 4096 registers |
| MP | map memory, 1024 registers |

These names are consistent with all other microprocessor and microcode literature.

The Infoton display is completely controlled by Midas. The upper seven rows display 14 of the microprocessors internal registers. The lower 8 rows can display any memory words from the memories listed above.

A address, one of 3 forms

i) symbol identifier, possibly followed by blank and octal increment (increment may contain leading + or - sign)

ii) memory identifier followed by blank and octal address

iii) octal address, presumed to be in instruction memory

V octal value

P screen position

The letter L(left) or R(right) followed by a decimal integer from 1 to 15(row)

F a file name.

If it contains no period, a default extension will be added, as specified for each command.

Octal values, addresses and increments may contain embedded blanks.

Rubout deletes the previously typed character and backspaces the cursor.

---

[1]These memories do not presently exist in Maxc1.

### 10.2. Commands

In the following, some commands have two forms, with and without a screen position field. In the form without a screen position field, the current screen position is used. Unless otherwise noted, all commands with a screen position field reset the current positiont to that specified in the current command.

| | |
|---|---|
| A= | prints out numeric value of A, in octal |
| : | single step |
| A: | single step at address A |
| A/<br>A,P/ | display contents of address A at given position |
| <cr><br>P<cr> | displays at given position the contents of next<br>higher address than currently displayed there |
| ↑<br>P↑ | displays at given position the contents of next<br>lower address than currently displayed there |
| <lf> | displays, at position below current position, contents of next higher address than currently displayed at current position. Resets current position one lower. |
| V←<br>V,P← | store octal value into the memory word addressed or register<br>named at the given screen position |
| A;B | insert a break point at given instruction memory address |
| A;K | remove a break point from given address |
| A;G | start processor at address A |
| ;P | continue running the microprocessor at the current microaddress (in IMA) |
| ;S | single step processor (same as :) |
| A;S | single step processor at address A (same as A:) |
| ;C | repaint the screen |
| F[1],...,F[k];R | load files F[1] to F[k]<br>(default extension ".MB") |
| 3,F;T | (default extension ".XX") take commands from file F until exhausted. May not be nested. |
| 4,F;T | (default extension ".ST") define /R file for subsequent patch calls to MICRO |

5,F;T (default extension ".MB") dump state of microprocessor onto file F, for subsequent reload using F;R. State dumped consists of the complete IM, SM, DM, LM, and RM memories but does ~!not! include the MP memory or the registers. The address symbols are dumped also. Reloading, the dump file takes about one minute ten seconds and it occupies about 52,000 characters on the Nova disk.

6,F;T (default extension ".SV") makes .EXEC DOS system call upon file F.

!F<sp><cr> (null default extension) is the same as 6,F;T except that F can contain spaces and slashes for constructing more elaborate .EXEC calls (e.g., "!NVIO.SV/B<sp><cr>" or "!POWER.SV OFF/M<sp><cr>"). The .SV extension is always required, and the trailing space nearly always.

<sp>TEXT<cr> will carry out microassembly of the TEXT using the file specified by the last 4,F;T as a /R file for MICRO. The text of the patch is appended to DBGPTCH and the binary is loaded. Two garbage files PTCH$$ (text of last patch) and PTCH$$.ST (/R file created by assembly) are also left.

### 10.3. Special Information

The cursor on the Infoton display will rise above the line when a command is in progress and drop down again when it is done.

Control-A will *stop* the microprocessor if it is *running*, but crash Midas back to the DOS EXEC if the microprocessor is *not* running (in which case you will have to restart by reloading the microprocessor). Midas can take 15 seconds to evaluate a new symbol near the end of a big microprogram so don't get impatient and type control-A. (Reloading from a big file takes over two minutes.) The cursor will rise above the command line when any command is in progress. If the command is a ;G or ;P, then the microprocessor will be running and control-A can be used to stop it. However, if you say SYMBOL;G, be sure you have waited the required 15 seconds for symbol lookup before typing control-A.

The *first* time you reference a symbol, Midas takes up to 15 seconds. Subsequent references are relatively fast (about 1/2 second).

Instruction memory addresses can only be displayed in the left column of the display and take up the full width of the screen.

When the microprocessor is stopped by control-A it will sometimes be possible to continue by ;P safely, but don't count on it. It is possible to continue safely from breakpoints except when the break occurs during the read portion of a read-modify-write memory reference or on either of the two cycles following a write, if the memory data register has not been loaded.

When Midas runs, it creates two temporary files called $$DBGE and $$DGBS. If you use the patch feature, PTCH$$, PTCH$$.ST, and DBGPTCH also get created. If you wish you may delete these when you return to DOS.

Do not attempt to examine main memory locations >177777 with Midas. Midas will go through the motions of examining and changing the addressed cell, but in fact the address will be truncated to 16 bits. All main memory addresses may be examined using ODT, as follows:

| | |
|---|---|
| !NVIO.SV/H <cr> | Starts NVIO but leaves Maxc halted. |
| NVIO | |
| :addr/contents | Examines main memory. |
| :M...OK. | Returns to Midas. |

# 11. MAXC2 MIDAS OPERATION

Midas is the loader/debugger used for the Maxc2 microprocessor. It has features for directly testing the Maxc2 hardware through the maintenance interface, for loading/dumping microprograms assembled by Micro, and for examining and modifying the storage in the microprocessor. It can control the microprocessor and memory power supplies, configure the memory, and enable/disable the various memory error correction-detection stuff. It also has a command for calling AltIO, the I/O program used by Tenex.

## 11.1. Starting Midas

To start Midas, simply say "Midas" to the executive or, more generally, "Midas com-file".

Midas command files have the extension ".Midas". Generally, there is one command file for each of the hardware diagnostics, with the same name as the diagnostic, e.g.:

| | |
|---|---|
| dgbasic.mb | the diagnostic; |
| dgbasic.midas | the command file; |
| midas dgbasic | to the Executive executes the command file. |

These command files load the diagnostic into the microprocessor and display various registers which are of interest when the microprogram is in use.

In addition, there are the following command files:

| | |
|---|---|
| midas/i | initializes (should be executed whenever any Midas files move or change, or when a new Alto Operating System is installed). |
| midas/r | continues with the symbol table and display saved at the last call to AltIO. |
| midas | simply fires up Midas. |
| midas tenload | loads the Tenex microcode. |
| midas tengo | loads Tenex microcode and calls AltIO to start Tenex. |
| midas mexecgo | loads Tenex microcode and calls AltIO to start MicroExec. |
| midas init | loads new Tenex microcode and sets checksums. |
| midas debug | loads some special symbols for use with "Constructed-Test" (see below). |

If you are already running Midas, and you want to switch microprograms, you can bug the "Run-program" menu item and then bug the name of the command file in the subsidiary menu which comes up. The "loader" menu item simply initializes a new Midas. "debug" is used to prepare for the "Constructed-test" stuff discussed later.

## 11.2. Midas Display

At the top of the Midas display are a number of register name-value menus. Below these are the name of the last microprogram loaded, the command comment line, the command menu, and the

input text line. When you move the mouse around, the menu item selected (if any) turns black. Note that mouse actions execute when you *release* the mouse button, so you can move the mouse around with the button depressed without causing damage. If the mouse position does not select any screen item, nothing happens when the button is released.

Register areas are of different sizes. Smaller areas are already filled in with various microprocessor registers when you fire-up Midas. The unused ones at the right and bottom left of the display are appropriate for 36-bit stuff, but only the bottom right items are large enough for IM (instruction memory) stuff.

To display a new item, type its name, move the mouse to one of the register name areas, and push-and-release the left (or top) mouse button.

If the command line is empty, the selected register area will be cleared when the button is released.

When you push the right (bottom) mouse button over a name area in which an address is displayed, a subsidiary menu appears as follows:

A+1 A-1

"A+1" increments the address, displaying the next location. "A-1" decrements the address.

Releasing the middle button over an address item shows alternate forms of printout on the command comment line. If the input text line is non-empty, it will first display that item.

Releasing the left button over a value item, evaluates the input text and stores the value (or 0 if no text typed) in the selected register. The input text is limited to octal numbers with interspersed spaces permitted for readability.

Releasing the middle button over a value item shows an alternate display of the value on the command comment line. The alternate for IMA, NPC, and STK values is the nearest IM address tag less-equal to the value+offset (the value is also put on the input text line, so you can examine that location in the display area if you want to). For DM, DM1, and DM2 words, the alternate is three IM addresses with offsets and one flag. For IM the alternate is a symbolic printout of the microinstruction.

Releasing the right button over a value item appends the text of the value to the input text line.

### 11.3. Midas Command Menu

For the command menu, all mouse buttons are presently equivalent (this may change). For most common commands, equivalent input text sequences carry out the same action, as given below.

The general philosophy on mixing keyboard and mouse button control is that, when possible, a command involving some typing is carried out completely at the keyboard, whereas commands involving mouse buttons are carried out completely with the mouse.

Many of the commands are executed in overlays. When these get executed, the register display will turn black (the code for overlays resides where the display bit buffers would otherwise be). During loading and during execution of command files, the display is turned off to make the machine run faster.

Many of the commands put up a succession of subsidiary menus with assorted options as discussed below.

The long-running commands normally display an "Abort" menu item. When this is bugged or when control-C is typed, the action terminates.

| Input | Keyboard | Menu Item | Comments |
|---|---|---|---|
| | ;Q | Exit | Quit to Executive. |
| File | | Read-Cmds | Execute command file (default extension ".Midas"). |
| | | Show-Cmds | Show equivalent command file text for selected menu items. |
| File | | Write-Cmds | Write subsequent commands on file. |
| | ;A | AltIO | Call AltIO with options (terminates command file). |
| | | Run-Program | Run selected microprogram (restricted use in command files). |
| Files | ;L | Load | Load .MB files. |
| Files | | LoadSyms | Load only addresses from .MB files. |
| File | ;D | Dump[2] | Dump compacted .MB file using the .MB file(s) of the previous load to control what's dumped. |
| File | ;C | Compare[2] | Compare microprocessor data to data specified in .MB file--compare file must have been created by Dump. |
| Addr | = | | Print value of an address (illegal in command file). |
| IMaddr | ;B | Break | Insert breakpoint. (Restarting from the breakpoint will succeed in all situations except when a memory write or read-modify-write has been given and MDR not yet loaded with the correct data or when the disk controller is active.) |
| [IMaddr] | ;K | Kill-Break | Remove break at address (at IMA if nothing typed). |
| [IMaddr] | ;G | Go[1] | Start at address (continue at IMA/NPC if nothing typed). |

| Input | Keyboard | Menu Item | Comments |
|---|---|---|---|
| [IMaddr] | ;P | Go[1] | Same as ;G (mnemonic "Proceed", provided for compatibility with Maxc1 Midas). |
| [IMaddr] | : | Step | Single-step at address (at IMA/NPC if nothing typed). See caveats on breakpoints above. |
| [IMaddr] | ;R | Repeat-Go[1] | Go at address, repeat endlessly after halts. |
| [IMaddr] | ;S | Repeat-Step[1] | Repeatedly step at address. |
| | | LMPEscan | Scan all local memories (IM, SM, DM, DM1, DM2) for parity errors. |
| | | Power-On[1] | Turn on power (see below). |
| | | Power-Off[1] | Turn off power (see below). |
| LDRaddrs | | Constructed-Test[1] | (see below). |
| | | Test[1] | Test register or memory (see below). |
| | | Field-Loop[1] | For scoping (see below). |
| | | Test-All[1] | Test everything (see below). |

[1] Requires preceding "TimeOut" command in command file.

[2] Requires confirmation with <cr>, "Y", or "." (or by preceding "Confirm" command in command file).

### 11.4. Keyboard

There are a number of characters which are legal symbol constituents in microprograms but which will cause trouble for Midas when they appear in addresses.

Lower case typein is converted to upper case by Midas, so avoid lower case characters in microprogram identifiers. I recommend writing microprograms with the shift-lock key depressed. Avoid "=" in identifiers. "+" and "-" are ok so long as the following character (if any) is a letter.

Typing ahead is legal until the character you type would cause execution of a command. After that Midas will flush input and blink at you until the current command finishes.

At the end of a command the input text typed for that command is displayed on the input text line. This text remains valid and can be used as the argument for another mouse action. However, if you type any character (except control-A or backspace), the old input will be flushed before inserting the new character.

Keyboard editting characters are as follows:

| | |
|---|---|
| control-A | delete last character |
| backspace | delete last character |
| control-Q | clear text line |
| del | clear text line |

Other special keyboard characters are as follows:

| | |
|---|---|
| control-C | aborts the current action |
| control-Z | aborts a command file |
| control-D | turns on the display |
| control-O | turns off the display |

The interrupt characters above are ineffective during loading, dumping, or comparing, which typically take between 2 and 15 seconds. Indefinite duration commands, such as "Go", "Test", etc. always monitor the keyboard, so control-C can be used to terminate them.

Control-Z, control-D, and control-O are intended for use during command files. However, these characters do not take effect until the command file executes a command such as "Go" which monitors the keyboard. There is no way to abort a command file and give control back to Midas safely except during a "Go" or other long-running command. This is not expected to be a problem because commands are executed quickly.

After interrupting a "Go" with control-C or control-Z, proceeding with ";P" or ";G" will succeed except when the conditions discussed earlier for breakpoints apply.

Although command menu items "Step", "Go", "Break", "Kill-Break", "Repeat-Step", and "Repeat-Go" are provided, you will normally find it more convenient to execute these from the keyboard using the alternate command characters.

### 11.5. Command Files

Command files have default extension ".Midas". They are normally executed by typing "Midas com-file" to the executive or by bugging the name of a command file from the subsidiary menu put up by "Run-program". However, you can also execute a command file by typing a file name and bugging "Read-cmds" in the command menu.

To find out what text should be put in command files, you can bug the "Show-Cmds" item in the command menu. This will cause the command file text for each command to be displayed on the command comment line as the mouse selects it (you don't have to execute the command to see the equivalent text). This text is complete except that the mouse button which executes the command isn't shown unless you depress the mouse button. Usually the text "L " precedes the text given by "Show-Cmds", indicating that the left (or top) button was pressed and released to carry out the command. "M " precedes a middle-button command and "R " a right-button (or bottom-button) command. The name of the command is followed by a blank, then the command line text, and finally a carriage return. To terminate "Show-Cmds", bug "Conceal-Cmds" (this only appears in the command menu when "Show-Cmds" is in progress).

You can prepare a command file (default extension ".Midas") by typing a file name and bugging "Write-cmds". This causes text for subsequent commands to be put on the file. When you are done with this, bug "Stop-Write-Cmds" to close the file. ("Stop-Write-Cmds" only appears in the command menu when a command file is being written.) You will probably want to edit out your goofs with Bravo after the command file is written.

For command files associated with big programs, the symbol table is not entirely in core, so you may want to sort the command-file addresses in reverse-alphabetical order to minimize symbol-table swapping. Currently, Midas has barely enough symbol table space to hold all of the Tenex microcode's addresses, so none of the symbol blocks have to be swapped. However, this may change if more features are added to Midas or to the Tenex microcode.

Command files can be nested several levels (limited by the size of sysZone which must be big enough to accommodate OpenFile and buffers for the command files already open). However, there are the following *restrictions*:

(1) "AltIO" terminates command files (i.e., upon return to Midas from AltIO the command file will not be continued).

(2) "Run-Program" is illegal except in the top level command file. ("Run-Program" resets Midas, then calls "Read-Cmds". This reset operation smashes the symbol table, the display, and the stack back to their initial state. Hence, if you were to execute "Run-Program" from a subsidiary command file, that command file would be continued, but the higher level ones would not, and the sysZone buffers for the higher level command files would not be released.)

Extra <cr>s can be used in a command file for punctuation, and ";" can be used at the beginning of a line or after the last character of a command to begin a comment. The comment is terminated by <cr>.

Since Midas builds a table of file pointers during its initialization, when you edit an existing command file or .MB file, you should reinitialize Midas by typing "Midas/I". When you add new command files or .MB files you should update the "Midas.Programs" file appropriately and do "Midas/I". The form of "Midas.Programs" is discussed later.

There are a number of commands which can never occur when Midas is run interactively, but which are useful in command files. These are as follows:

| **Text Arg** | **Menu Item** | **Comments** |
|---|---|---|
| Octal no. | SkipVEql | Skip the next command if the selected value menu item equals the value of the text arg. |
| Octal no. | SkipVGr | Skip the next command if the selected menu item's value is greater than the text arg. |
| Octal no. | SkipVLs | Skip the next command if the selected menu item's value is less than the text arg. |
| Octal no. | Skip | Skip N following commands, where N is the 16-bit value of the text arg. |

| | | |
|---|---|---|
| Octal no. | BackSkip | Reset to byte 1 of the command file, then skip. |
| Octal no. | Return | Return out of current command file and skip N commands in the calling command file (if any), where N is the 16-bit value of the text arg. |
| | DisplayOn | Turn on the display, so that effects of subsequent commands can be observed. The display is initially off for a command file. |
| | DisplayOff | Turn off the display. |
| Octal no. | TimeOut | Argument is a 32-bit octal number of milliseconds after which to abort the immediately following command, if it has not finished by then. This is intended for use before "Go", which might hang indefinitely. If the timeout occurs Midas will skip the command after the "Go". TimeOut also turns on the display, which is necessary because the machinery which checks for timeouts is only active with the display on. |
| | Confirm | Supply confirmation for the command which follows (which should be one of the commands requiring confirmation). |
| File name | OpenOutput | Open an output file (default extension ".Report") on which text can be written. |
| | CloseOutput | Close the output file. |
| [text] | WriteMessage | Write the contents of the input text buffer on the output file. Note that if any text follows the WriteMessage, that text up to but not including the <cr> is what gets written. However, if <cr> immediately follows WriteMessage, then the contents of the input text buffer left by the previous command are what gets written. "~" is translated into <cr>. |
| text | Show-Error | Display the text arg on the command line, turn on the display if it was off, and query with "Abort" and "Continue" menu items. |

The "right-button" value action allows the current values of any register to be appended to the input text buffer. This can be used in conjunction with WriteMessage to output various parts of the machine state to the output file.

### 11.6. Loading Programs

The "Run-program" command is a short method of executing the most common command files--those which load a microprogram and fix up the display for that program.

You can also load a program by typing a file name (default extension ".mb") and bugging "Load" in the command menu. However, this should rarely be necessary because all of the microprograms appear in the menu put up by "Run-program", which additionally reinitializes Midas.

It is a poor idea to do a "Load" when another microprogram has already been loaded because this will merge symbols from the new load into those already in the symbol table. In addition to being very slow, this may overrun the size of the symbol table on the disk (no provision is made for expanding the symbol table beyond its maximum size).

It is also a good idea to assemble microprograms as a single .MB file. Although Midas can load multiple .MB files (typed as a list separated by commas), this will cause additional fragmentation of the symbol table and thrashing while blocks are swapped in and out.

These recommendations follow because Midas takes advantage of alphabetical address ordering in .MB files to pack its symbol buffers nearly full. But when subsequent files are loaded the symbol buffers will be fragmented to about half-full, symbol buffer swapping will result, and the symbol searches will be longer.

### 11.7. Dumping Microprograms

After assembling a microprogram, it is a good idea to load and dump it from Midas. Dumping deletes all forward reference fixups left by Micro and compacts both data and addresses to use less disk space and load more quickly later. Also, undumped .MB files cannot be used with "Compare" because of forward references.

Note that only those memory words loaded by the original microprogram are dumped, so you cannot patch unused locations, dump the program, and expect the patches to survive.

### 11.8. Tenex Microcode

When a new release of the Tenex microcode is made, it must be loaded into the microprocessor, the checksums set, and then dumped onto SYS2. To carry out this initialization, type "Midas init" to the executive.

After dumping, SYS2;C can be used to compare the microstore against its correct contents whenever the checker program (INIT;G) fails. This is a good way of locating bad storage chips after a crash or verifying that the storage is OK before loading some other microprogram.

### 11.9. Power On-Off

The "Power-On" menu item allows the microprocessor, port, and memory to be powered on and tested in various ways. The first "Power-On" subsidiary menu shows the following:

| | |
|---|---|
| Processor | Turn on processor and port power only. |
| Memory | Turn on memory and port power only and configure memory. |
| Both | Turn on port, processor, and memory cabinets' power. |
| Test-Port-&-Processor | Repeatedly execute the maintenance interface operations for turning on port and processor power (for debugging power control hardware). |
| Test-Memory-Cabinet | Query for a memory cabinet number, then repeatedly execute the I/O instructions for powering on the memory cabinet. |

After the processor is turned on, the local memories (IM, SM, DM, DM1, DM2) are zeroed to prevent parity errors from occurring.

When "Memory" or "Both" is bugged, another subsidiary menu will come up showing the various options for enabling/disabling error correction, core-zeroing, and FER on memory errors. For normal use by Tenex, let the default action take place for all of these options, and immediately bug the "do-it" menu item.

Power on takes about 15 seconds because the power supplies take a while to stabilize. After power-on the interrupt system is cleared and the flag register is zeroed.

The right-most four nine-bit bytes in CONFIG (appears on the Midas display) control the memory configuration as follows:

| | |
|---|---|
| bits 0-8 | represent quadrant J |
| bits 9-17 | represent quadrant K |
| bits 18-26 | represent quadrant L |
| bits 27-35 | represent quadrant M |

A "1" in the byte enables cabinet 0, "2" cabinet 1, "4" cabinet 2, ..., "200" cabinet 7. Hence, CONFIG might be set as follows:

| | |
|---|---|
| 3003003003 | All four quadrants active in cabinets 0 and 1 (would result in 256K four-quadrant configuration) |
| 17017017017 | All four quadrants active in cabinets 0-3 (would result in 512K four-quadrant configuration) |
| 3000003003 | Quadrant K disabled, but cabinets 0 and 1 active (would result in 128K two-quadrant configuration avoiding quadrant K) |

Memory power-on does a catastrophic memory reset. This may occasionally clobber memory words, but this complete reset has been required to unhang the memory on several occasions, due to problems which have not been fixed.

The first "Power-Off" subsidiary menu is analogous to the "Power-On" menu. Note that the port power will remain on unless you bug "both" because neither the memory nor the processor is operable without the port power being on.

Please be mindful of the following *terrible danger:* **The microprocessor, port, and memory power must be turned off before powering down the Alto.** When this rule was violated on two occasions, numerous Alto MI chips were clobbered, apparently due to power transients from the cable.

To make repairs to the microprocessor, it is sufficient to turn only the processor power off, provided that neither the PMEMI, KMEMI, nor PMAINT cards are disturbed. For all other maintenance, power down "Both". If you want to remove the PMAINT card, power off the Maxc processor and memory first, then the Alto disk, and finally, the Alto processor.

### 11.10. Testing Through the Maintenance Interface

"Test", "Constructed-Test", and "Test-All" allow the microprocessor to be tested from the Alto through the maintenance interface. Data patterns for the test are determined from the first subsidiary menu, as follows:

| | |
|---|---|
| ZEROES | All-zeroes data |
| ONES | All-ones data |
| CYC1 | 36-bit word of all zeroes with a single-one bit cycled left one position each iteration |
| CYC0 | 36-bit cycled zero in word of ones |
| RANDOM | 48-bit random numbers |
| SEQUENTIAL | 0, 1, ..., sequential numbers |
| ALTZO | Alternating all-ones and all-zeroes patterns |
| ALT-SHOULD-BE | Alternating contents of SHOULD-BE with its ones-complement |

Testing is controlled/described by five addresses on the display as follows:

| | |
|---|---|
| LOW-HIGH | (For memory tests only) contains in bits 0-17 the lowest memory address tested and in bits 18-35 the highest address tested. Memory tests begin at the low address and sequentially test each address up to the high address, then loop. |
| LOOP-COUNT | The number of successful iterations of the test prior to failure or prior to aborting from the keyboard or with the mouse. |
| SHOULD-BE | What the data should have been. |
| DATA-WAS | What the data actually was. |
| BITS-CHECKED | Mask of bits checked (see below). |

LOW-HIGH delimits the maximum address range. If the memory you select has fewer words than HIGH, then the maximum for that memory is used. This means that the last address tested for a memory test is as follows (assuming LOW-HIGH is left at its normal maximum value):

| | |
|---|---|
| IM | LOOP-COUNT mod 10000 (octal) |
| SM | LOOP-COUNT mod 1000 |

| | |
|---|---|
| DM | LOOP-COUNT mod 1000 |
| DM1 | LOOP-COUNT mod 1000 |
| DM2 | LOOP-COUNT mod 1000 |
| RM | LOOP-COUNT mod 40 |
| LM | LOOP-COUNT mod 40 |
| STK | LOOP-COUNT mod 14 |

"Test-All" automatically loads BITS-CHECKED with a full-sized comparison mask prior to testing each item. It tests each register 250 times and makes 4 passes through each memory. It is a good idea to run "Test-All" whenever the hardware is in a suspicious state.

"Test", after showing the data-pattern menu, shows a menu of register and memory names, and executes a test of the one you select until the test fails or you halt the test from the keyboard. "Test" masks the current contents of BITS-CHECKED with the maximum-sized mask for the register or memory being tested. If you previously tested a small register, then you have to manually load BITS-CHECKED with a full-sized mask before testing a big register. If you don't want to check all the bits in a register, then clear the bits you don't want to check in BITS-CHECKED.

"Constructed-test" should only be used when the "debug" command file has been loaded. This requires a sophisticated understanding of the hardware and of the innards of Midas and is not recommended for novices. Midas makes use of a number of microinstructions while reading, writing, starting, and stopping the microprocessor. "Constructed-test" allows these microinstructions to be executed in non-standard sequences to beat on particular hardware problems through the maintenance interface. When using "Constructed-Test", a list of LDR addresses is typed (see DEBUG.MC for the instructions) separated by commas. If only one LDR address is typed, the maintenance interface's BR register is loaded once with the selected data pattern, then the LDR instruction is repeatedly executed for a scope loop. When two, three, etc., up to six LDR addresses are typed, a test loop occurs whereby BR is loaded with the next data pattern, the list of instructions is executed, and then BR is read back and compared against the data under control of BITS-CHECKED. The loop stops when (data-read-back xor data-sent-out) & BITS-CHECKED is non-zero.

"Field-Loop" exercises signal decoding for particular fields of the microinstruction for scope loops. A microinstruction is fabricated from a no-op microinstruction in which the field selected from the first subsidiary menu is replaced by various values. The second subsidiary menu allows the value in the selected field to be incremented, decremented, and shifted.

## 12. OPERATING TENEX MICROCODE FROM MIDAS

**A.** Reset is accomplished by typing

21;G

to Midas. The effect of this is to put the machine in monitor mode, to clear the interrupt system, the APR, the processor flags and the disk system. However, Nova peripherals aren't affected, nor are the accumulators or PC affected. If NVIO or AltIO is called after RESET the processor will start at its starting address (pointed to by absolute location 7).

**B.** Starting at an arbitrary PDP-10 location n is accomplished by changing the contents of PC (which is on the display) to n and then typing

| | |
|---|---|
| REMAPPC: | (Midas types out the value m of REMAPPC) |
| 6,NVIO;T | (*Maxc1 only*) |
| "AltIO", "Do-It" | (*Maxc2 only*) |

**C.** The PDP-10 accumulators are LM 0, LM 1, ..., LM 17 and can be examined and changed in the usual way from Midas.

**D.** The PDP-10 flags are in the left-most 13 bits of the F-register on the display. If you change these from Midas, don't change the other 23 bits of F.

These and other interesting bits of F are as follows:

| Bit | Definition |
|---|---|
| 0 | Overflow |
| 1 | Carry 0 |
| 2 | Carry 1 |
| 3 | Floating overflow |
| 4 | Byte increment suppress |
| 5 | User mode |
| 6 | PARC mode (replaces PDP-10 user I/O mode) |
| 7 | Call from monitor (see JSYS, UMOVE, and Pager addendums to PDP-10 System Reference Manual) |
| 8-10 | Machine mode (0 = PDP-10, 1 = Byte Lisp) |
| 11 | Floating underflow |
| 12 | No divide |
| 13 | Pushdown overflow |
| 14 | XCT0 |
| 15 | XCT1 |
| 16 | XCT2 |
| 17 | XCT3 |
| 18 | Incompatible |
| 19 | PI system is active |
| 20 | PI cycle in progress |
| 21 | MONALT - Temporary flag used by map loading microcode |

| | |
|---|---|
| 22 | THIRDPT - Temporary flag used by map loading microcode |
| 23 | Unused |
| 24 | TTYBSY - console teletype output busy |
| 25 | LOGF - enables main loop "LOGI", "BRKI", or "TRACEI" path |
| 26 | PICYCLE |
| 27 | CUM |
| 28 | MICRO |
| 29 | IENABLE |
| 30 | Unused |
| 31 | NOVA - Nova/Alto has left an interrupt request |
| 32 | K |
| 33 | J |
| 34 | H |
| 35 | G |

E. The interrupt system state is given by the following:

PISTAT[29,35] have 1's when interrupts are in progress on the corresponding PI levels[1,7]. PISTAT[22,28] have 1's when interrupts are disabled on the corresponding PI levels.

SM 600 to SM 643 contain in bits 12-35 the interrupt locations corresponding to devices 0 through 35. The device assignments are in Maxc document 11.

SM 644 to SM 652 are the interrupt-enabled bit tables for priority interrupt levels 1 through 7. 1's in each word indicate that the corresponding device (bit 0=device 35, bit 35=device 0) is enabled for interrupts at that level.

MICINTS contains 1's for each device which has requested an interrupt. (However, OVF, FOVF, and PDOVF are recomputed from F for each instruction, so their state in MICINTS isn't important.)

F. The microcode consistency "Checker" uses four variables CSUMD, CSUM0, CSUM1, and CSUM2 which contain checksums for all constants in the microprocessor's IM, DM, DM1, DM2, and SM memories. CSUMD is an overall checksum (for detection of errors) while CSUM0-CSUM2 contain 18 6-bit bytes each of which is a checksum in a Hamming code. The checker computes the address which is clobbered assuming only a single word is wrong.

When newly assembled microcode is loaded for the first time on Maxc1, these checksums are computed by running CHECKER ("25;G" in the TENLOAD command file) and entering the values left in LM 11-LM 14 into the PATCHES file for CSUMD-CSUM2. On Maxc2, the checksums are computed and dumped automatically by "Midas Init". Subsequent runnings of the TENLOAD, MEXECGO, and TENGO command files check the newly loaded system against these constants, and crash on errors. Also the CHECK JMC does this (executed during Tenex initialization).

## 13. INTERPRETING CHECKER FAILURES

The microcode Checker is normally run whenever Tenex is started or restarted, and may be run manually by issuing the Midas command "25;G". Checker failures are characterized by:

a) IMA = 20

b) STK 0 = RETN (type "RETN=" to verify this)

c) a value in LM 10 between 0 and 77777.

To interpret a Checker failure, the following guidelines are offered:

a) If LM 10 contains 0, all SM, DM, DM1, DM2, and IM registers containing constants were verified to be correct, but some LM or RM constant or other processor operation failed. See the microinstruction at [STK 0]-1 in the Checker listing to determine what's wrong.

b) If LM 10 does not contain 0, it contains the logical "or" of all addresses containing incorrect values, where

| | |
|---|---|
| 00000-07777 | Instruction memory (IM) bits 0-35 |
| 10000-17777 | Instruction memory (IM) bits 36-71 |
| 20000-23777 | Dispatch memory (DM) 0-3777 |
| 24000-24777 | Scratch memory (SM) 0-777 |

Since these memories are composed of chips which span 400 or 2000 (octal) bits and since the normal failure mode is complete chip failure, it will seldom be true that the number in LM 10 is the address of a single failure. For total chip failures, the logical "or" of the addresses will wind up in LM 10. Note that on the old bipolar boards, chips are interleaved so that a chip hits every fourth address. The value in register LPGRT3 is the complement of the wrong bits in the word if there was only a single error.

c) On Maxc2, one can obtain additional information by invoking "LMPE-Scan" to find the bipolar memory locations containing parity errors, and by invoking "Compare"[1] to compare the current memory contents with the file from which the control memory was loaded. (Note that it is normal for "Compare" to find one error at location IODEND.)

d) Run the DGIMH and DGIML micro-diagnostics for further error information.

---

[1] "Compare" (Ref. Section 2, par. C.3.) produces a list of errors on file Midas.Error. From the Alto executive issue the command "Type Midas.Errors". The number associated with each error is a 36 bit word numbered from left to right bit 0 through bit 35. These bit numbers correspond to the bit numbers identified in the bipolar memory card chip maps (Figures 1, 2 & 3).

## 14. USING MICRO-EXEC

Micro-Exec is a stand-alone Maxc program used to maintain and start up the Maxc Tenex system. A few overall remarks on its structure will be helpful before describing the specific facilities available.

Micro-Exec is normally loaded by the MEXECGO Midas command file (see Section 5) from Maxc disk unit A. Micro-Exec loads into the high portion of the first 128K of memory, starting at 350000. All references to "core" in Micro-Exec commands refer to the range 20-to-347777. (The program which boots in Micro-Exec runs in 347000-347777 and saves the previous contents of the ACs in 347760-347777.)

Micro-Exec has a sophisticated command interpreter which allows editing, questioning, prompting for parameters -- what we in the biz call "Dwimified-Middle-English". Commands take the form of a sequence of words separated by periods. E.g.:

```
        initialize.disk.pack
or
        initialize.tape.
```

At any point in typing a string, you may type <escape> and the interpreter will complete as much as is unambiguous. Similarly, at any point you may type "?" and all the possible completions of that command will be listed. Once you are familiar with commands, you will find it convenient to abreviate them by entering the first letter of each word followed by a space. For example, I D P is an abbreviation for "initialize.disk.pack".

Many of the commands require parameters before the command execution begins. Typing an escape will prompt you with a brief description of the parameter (e.g., (octal number) or (pack number)). Typing escape a second time will furnish the default value of that parameter if there is one. Some commands require confirmation (with carriage return or period).

Micro-Exec is capable of running in user mode under Tenex as well as stand-alone. Micro-Exec may be accessed under Tenex by running the subsystem MEXEC, and it requires enabled "wheel" or "operator" privileges. All commands may be executed either stand-alone or in user mode except where noted otherwise.

### 14.1. Tenex Disk Structure

Finally a few words on the disk structure of the Maxc-Tenex system will be helpful for the following command descriptions. A disk configuration is defined by establishing a correspondence between (1) logical units (Tenex), (2) pack numbers, and (3) disk drives (i.e., controllers). The current disk configuration is usually the default configuration on the current Micro-Exec save area (usually area 1 on physical unit 0). Logical units are identified by digits 0 thru n-1 (for an n-pack Tenex). Packs are labeled with octal numbers starting at 100. Drives are labelled alphabetically starting with A. For example, the "print.disk.configuration" command might type:

```
Number of units: 2
----------Unit--------Pack Number
0.          B           102
1.          A           107
```

At present there are 20 "Save Areas" allocated in the disk structure, but outside the Tenex file system. Each save area consists of four track cylinders (i.e., 240 (decimal) pages). A directory of save areas is posted on the control room wall.

A mechanism has been provided for maintaining a simple file system on a single save area, for storage of small programs such as PDP-10 diagnostics. At present, save area 2 is assigned permanently for this purpose. All instances of the word "program" in command names refer to programs stored in this manner. Sometimes disk packs are moved from one drive to another, so that drive can be freed for maintenance. When this is done, the procedures for automatically starting Tenex or booting the Micro-Exec may not work and some of the issues are discussed here. First, the command files which start Micro-Exec or Tenex from Midas expect to find Micro-Exec on save area 1 of disk drive 0 and Tenex on save area 0 of disk drive 0. If the pack containing these areas is moved to some drive other than 0, these command files won't work. In this case, you must boot Micro-Exec by hand from NVIO/AltIO rather than relying on the MEXECGO or TENGO Midas command files to do this for you (see the chapters on NVIO/AltIO for how to do this). After starting up Micro-Exec (and setting the disk configuration, if necessary), you can do a "Go" command to start Tenex.

Secondly, the various methods for starting Tenex from Micro-Exec require that Micro-Exec know the correct disk configuration. If the configuration has changed since Micro-Exec was written on the same area, the configuration must be correctly entered into Micro-Exec ("Set.Disk.Configuration" discussed below) before starting Tenex. "Write.Micro.Exec.to.area.1" can be used to store Micro-Exec on its same area with a correct disk configuration.

### 14.2. Micro-Exec Command Descriptions

In the following descriptions, a (C) means confirmation is required. In commands that read from or write to disk, the number following the command indicates the error retry count used; "default" means that the retry count may be overridden by the count specified in the "set.disk.error.retry.count" command. If not specified, the retry count is 20 and is unaffected by the "set.disk.error.retry.count" command.

brief
: Disables extended typeout of disk error status bits.

check.next.tape.file <magtape unit number> (stand-alone only)
: Checksums next file on tape and types entry point on console.

clear.core (C)
: Zeros locations 20 through 347777 and 400000 through 777777.

compare.disk.packs <pack$_1$> <pack$_2$>
Compares the contents of the specified packs and reports discrepancies (usually used after a pack-to-pack copy).

copy.pack.to.pack <pack$_1$> <pack$_2$> (C) (default 20)
Does a bit-by-bit copy of data from first pack onto second pack.

copy.quadruple <pack$_1$> <pack$_2$> <pack$_3$> <pack$_4$> (C) (stand-alone only)
Simultaneously copies the first pack onto the second pack and the third pack onto the fourth pack.

ddt
Enters DDT for debugging Micro-Exec. Re-enter Micro-Exec by 20$G.

debug.disk.controller <unit$_1$> <unit$_2$> (C) (stand-alone only)
Permits checking out the disk controller hardware without using a real disk. This requires connecting the write cable of <unit$_1$> to the read cable of <unit$_2$> and connecting a pulse generator to simulate sector pulses.

dismount.auxiliary.pack
Removes auxiliary pack from disk structure.

dump.core.to.area <area>
Dumps locations 20 through 347777 on <area>. Registers are saved at 347760 when Micro-Exec is booted in. This command is used to save crashes for later analysis.

dump.core.to.file <filename> (user mode only)
Dumps locations 20 through 347777 on <filename> (default extension .SAV).

dump.program <program name> (C)
Dumps the (small) program presently in core onto the current program save area (usually 2) with the given name (5 or fewer characters). If a program already exists by that name, it is overwritten. The program in core should conform to PDP-10 conventions: the LH of location JOBCOR (133) should contain the highest location loaded with code, and the RH of JOBSA (120) should contain the starting address.

dump.save.areas.to.auxiliary.pack (C)
Copies all the save areas from the currently configured file system onto the auxiliary pack. The save areas are arranged as if the auxiliary pack constituted a one-pack file system; hence, the last 4*20 = 80 (decimal) tracks on the pack are overwritten.

eddt
Enters Exec DDT if Tenex is in core. Re-enter Micro-Exec by typing 20$G.

erase.program <program name>
Deletes the named program on the current program save area.

exercise.disk.random <pack> (default 0)
Reads pages at random on the specified pack, reporting any device-detected errors (checksum, etc.) but not actually looking at the data. The pack is not written on.

exercise.disk.specific <pack> (default 0)
Prompts for a list of disk addresses, which it then cycles through repeatedly, reading each specified page as explained above.

find.chip.for.address <address> <bit designator>
Translates the supplied memory address and bit number into a physical memory chip location. <address> is a 21-bit octal number, and <bit designator> is one of "Bit n", "Tag n", "Check n", or "Parity" (abbreviations allowed), where n is a decimal number. Data bits are numbered 0-35, tag bits 0-3, and check bits 1, 2, 4, 8, 16, and 32.

go (stand-alone only)
Executes a "test.memory.fast", then loads Tenex from area 0 and starts it at SYSGO1.

goto <address>
Starts whatever is in memory at the specified address.

initialize.area.for.programs (C)
Initializes the "directory" of the current program save area (usually 2).

initialize.disk.pack (C) (default 0)
Initializes the auxiliary pack by: (1) writing headers, using the pack number entered in the mount.auxiliary.pack command, (2) writing random data on the entire pack, and (3) checking the data and printing discrepancies. An error retry count of zero is used throughout. Pages in which errors are detected are recorded in the bad spot list on page zero of the pack.

initialize.tape <magtape unit> (C) (stand-alone only)
Rewinds tape to load point, writes a boot header, and waits at the end of the boot header.

install.new.micro.exec (C)
Transfers a new version of Micro-Exec (loaded via a previous "load.dump.from.area") to its runtime location, and starts it up.

load.dump.from.area <area>
Loads core from specified area (inverse of dump.core.to.area).

load.dump.from.file <filename> (user mode only)
Loads core from specified file (inverse of dump.core.to.file).

load.program <filename>
Loads into memory the specified program from the current program save area (usually 2).

load.save.areas.from.auxiliary.pack (C)
Loads all save areas onto the currently configured file system from the auxiliary pack, which must previously have been written by dump.save.areas.to.auxiliary.pack.

loop.on.specified.disk.page <pack> <track> <head> <sector>
Continuously reads the specified disk page ignoring errors (useful for disk tune-ups). The bell is rung for each error detected.

mount.auxiliary.pack <unit> <pack>
Identifies auxiliary (i.e., extra-Tenex) pack to disk configuration. Certain commands such as initialize.disk.pack are valid only for the auxiliary pack. Note that there can be at most one auxiliary pack known to Micro-Exec at a given time. In user mode, one should be careful to "dismount.auxiliary.pack" before leaving Micro-Exec to avoid confusion.

print.disk.configuration
Prints disk configuration in the format illustrated above.

print.disk.error.count
Prints the total number of disk errors (both recovered and unrecovered) since Micro-Exec was started.

print.program.directory
Prints the name and size (pages) of each program in the current program save area (usually save area 2).

quit (user mode only)
Returns control to the Tenex Exec.

read.specified.disk.page <pack> <track> <head> <sector> (default 20)
Reads specified page into buffer at location BUF.

read.tape.to.core <magtape unit> (stand-alone only)
Reads tape into core starting at location 20.

read.tenex.from.area <area>
Loads Tenex from specified area and checks its "fingerprint".

read.tenex.from.file <filename> (user mode only)
Loads Tenex from the specified file (default extension .SAV).

read.tenex.from.tape <magtape unit> (stand-alone only)
Loads Tenex from tape and checks its "fingerprint".

reset.disk.error.count

rewind.tape <magtape unit> (stand-alone only)

run.diagnostic.program <program name>
Loads and starts the named program from the current program save area, assuming that the program conforms to the conventions for controlling PDP-10 diagnostics. The diagnostic is run for one complete pass, and then control returns to Micro-Exec.

run.program <program name>
Loads and starts the named program from the current program save area.

scan.disk.pack.for.errors <pack> (default 0)
Scans pack, counting soft errors, and reporting non-recoverable errors to console.

set.disk.configuration (stand-alone only)
Sets disk configuration; asks for parameters via the same format used in print.disk.configuration.

set.disk.error.retry.count
Overrides the default disk error retry count in certain commands.

set.disk.timeout <decimal number> (stand-alone only)
Sets the disk timeout interval (seconds), at the end of which Micro-Exec will report that it has timed out.

set.program.save.area <area>
Declares the program save area to be used for subsequent program operations ("run.program", etc.) Default area is 2.

silent (C)
Completely turns off disk error typeouts (useful for setting up scope loops for hardware debugging).

start.tenex.from.area <area> (C) (stand-alone only)
Reads Tenex from the specified area and starts it at SYSGO1.

test.memory.fast (stand-alone only)
Runs a quick (~30 second) memory checkout and summarizes errors found. Also builds a map of bad memory for communication to Tenex.

test.memory.slow (stand-alone only)
Runs a thorough (~8-minute) memory test and summarizes errors found.

test.memory.verbose (stand-alone only)
Same as "test.memory.slow", but errors are printed out individually rather than only summarized.

test.memory.write.fast (stand-alone only)
Undoes the effect of test.memory.write.slow.

test.memory.write.slow (stand-alone only)
Sets a mode flag that causes subsequent runs of the memory test to write data into memory using a more conservative (but slower) method than usual. Try this command if the memory test crashes due to "fatal error from wrong place in code".

verbose
Reinstates full disk error typeout.

verify.disk.test.pattern (default 0)
Reads the auxiliary pack and verifies the data written by a previous "initialize.disk.pack" or "write.disk.test.pattern", reporting any discrepancies (up to a maximum of 5 per page).

write.core.to.tape <magtape unit> (stand-alone only)
Writes core from 20 to 347777 onto tape.

write.disk.test.pattern (default 0)
Writes random data on all pages of the auxiliary pack, then rereads and verifies it, reporting any discrepancies (up to a maximum of 5 per page).

write.micro.exec.to.area <area> (C)
Writes a bootable (via NVIO "B" and "S" commands) copy of the currently running Micro-Exec to the specified area. (Ordinarily Micro-Exec is kept on save area 1.)

write.micro.exec.to.tape <magtape unit> (stand-alone only)

write.specified.disk.page <pack> <track> <head> <sector> (C) (default 20)
Writes page from memory location BUF to specified disk page.

write.tenex.to.area <area> (C)
After checking "fingerprint", writes Tenex core image to specified area.

write.tenex.to.file <filename> (user mode only)
After checking "fingerprint", writes Tenex core image to specified file (default extension .SAV).

write.tenex.to.tape <magtape unit> (C) (stand-alone only)
After checking "fingerprint", writes Tenex core image to specified tape.

A more compact summary of Micro-Exec commands is given in the following table:

### 14.3. Micro-Exec Command Summary

*Disk Configuration Commands*

| | |
|---|---|
| set.disk.configuration | print.disk.configuration |
| mount.auxiliary.pack <unit><pack> | dismount.auxiliary.pack |

*Disk Pack Copy and Compare Commands*

| | |
|---|---|
| initialize.disk.pack | compare.disk.packs <pk1><pk2> |
| copy.pack.to.pack <pk1><pk2> | copy.quadruple <pk1><pk2><pk3><pk4> |
| scan.disk.pack.for.errors <pack> | |

*Save Area and Other Commands*

| | |
|---|---|
| dump.save.areas.to.auxiliary.pack | load.save.areas.from.auxiliary.pack |
| dump.core.to.area <area> | load.dump.from.area <area> |
| write.micro.exec.to.area <area> | install.new.microexec |

goto <address>
clear.core
ddt
eddt
read.tenex.from.area <area>
write.tenex.to.area <area>
start.tenex.from.area <area>
go

*Program Save Area Commands*

set.program.save.area <area>
initialize.area.for.programs
dump.program <program name>
erase.program <program name>
print.program.directory
run.diagnostic.program <program name>
run.program <program name>

*Disk Test Commands*

reset.disk.error.count
set.disk.error.retry.count
print.disk.error.count
set.disk.timeout <decimal number>
write.disk.test.pattern
verify.disk.test.pattern
exercise.disk.random <pack>
exercise.disk.specific <pack>
brief
verbose
debug.disk.controller <unit1><unit2>
silent
read.specified.disk.page <pack><trk><hd><sec>
write.specified.disk.page <pack><trk><hd><sec>
loop.on.specified.disk.page <pack><trk><hd><sec>

*Memory Test Commands*

find.chip.for.address <address><bit designator>
test.memory.fast
test.memory.slow
test.memory.write.fast
test.memory.write.slow
test.memory.verbose

*Tape Unit Commands*

initialize.tape <unit>
rewind.tape <unit>
read.tape.to.core <unit>
write.core.to.tape <unit>
read.tenex.from.tape <unit>
write.tenex.to.tape <unit>
write.micro.exec.to.tape <unit>
check.next.tape.file <unit>

*User Mode Commands*

dump.core.to.file <filename>
load.dump.from.file <filename>
read.tenex.from.file <filename>
write.tenex.to.file <filename>
load.program <filename>
quit

## 15. DMPLD

*Maxc1 only.* (The "load" function of DMPLD is implemented by the AltIO "Load" command on Maxc2.)

DMPLD is a Nova program[1] which can transfer stand-alone PDP-10 programs between Nova files and Maxc's memory. At the present time, it is used only for loading PDP-10 diagnostics to assist in microcode debugging.

### 15.1. DMPLD Operation

To load a PDP-10 .SAV program when you are in Nova DOS type

DMPLD<cr>

and answer the LOAD FROM FILE question. Type rubout and try again if you make a mistake. This zeroes the first 32K of main memory before loading.

If you are in Midas, type

6,DMPLD;T

Normally you will have to prepare locations 3, 4, and 7 before running the program. To make these changes, proceed as follows:

DMPLD<cr>
LOAD FROM FILE: AUXD<cr>
NVIO/H<cr>
3/ 0 nnnnn<lf>
4/ 0 nnnnn<cr>
7/ 0 nnnnn<cr>
M...OK.
DEB DMPLD<cr>
DMPLD%
DUMP$R
DUMP TO FILE: AUXX<cr>

The file will typically grow by a substantial amount when this is done because DMPLD does not compress out zeroes when dumping.

DELETE AUXD<cr>
RENAME AUXX AUXD<cr>

---

[1] At present, the only way to create such a file on the Nova disk is to dump it from Maxc memory using DMPLD. When Maxc was originally being bootstrapped, such files were created using a program called Mtape, which read Tenex "Mini-Dumper" tapes mounted on a 7-track drive which no longer exists.

If you want to dump a PDP-10 program you have created in core, you must exit back to DOS (by control-A or ;X to Midas). Then dump the first 32K of main memory by typing

```
DEB DMPLD
DMPLD%
DUMP$R
```

The program will then ask DUMP TO FILE: and you should type a file name. If you make a mistake type rubout and try again.

### 15.2. Required Format for Standalone PDP-10 Programs

DMPLD requires the program to be a .SAV file which resides in the low 32K of main memory. DMPLD currently zeroes the low 32K of core before loading the program.

The interprocessor communication locations must be set up with pointers as follows:

| | | |
|---|---|---|
| 3/ | MTBS | Magtape, IMP, MCA, etc. 300-word block (octal; may grow later) |
| 4/ | DLSBS | Data line scanner 104-word block |
| 7/ | STADR | Program starting address |

Note that these are absolute main memory locations in the "shadow" of the accumulators, so they must be initialized in one of three ways:

(1) By hand from Midas or ODT
(2) By mapping page 0 to a different virtual address
(3) By using the JMC's for addressing absolute main memory addresses.

The program must perform an I/O reset (CONO APR,200000) *before* carrying out input/output to any Nova related peripherals and *after* initializing MTBS and DLSBS as described above.

# 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.<br><br>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 | |

## 17. WRITING A NEW 10SYS TAPE

*Maxc1 only.* If any system files have been changed, they should exist on the Nova disk prior to beginning the following procedure. It is assumed that all the normal system files are on the disk (if not, load them from the current 10SYS tape first).

A. Clean up the disk as much as possible by deleting scratch files and other extraneous junk.

B. Mount the new 10SYS tape on unit 0, with a write ring in it. (You must use magtape unit 0 for 10SYS tapes since DOS will not handle any other drive.)

C. Type:

```
XFER TBOOT.SV MT0:0; XFER MAXCSYS.SY MT0:1 <cr>
DUMP/A/V MT0:2 <cr>
```

D. Check that everything is written ok by typing:

```
LOAD/A/V MT0:2 <cr>
```

"File already exists" should be printed for every file.

E. Remove write ring. Label tape jacket as follows:

```
0 TBOOT.SV
1 MAXCSYS.SY
2 *.*
<current date>
 ...
Current 10SYS
<tape number>
```

F. Change the label on the old 10SYS tape to read "Obsolete 10SYS <current date>".

## 18. RECOVERY FROM CHECKDSK ERRORS

As explained in Section 6, when Tenex is restarted, whether initially or due to an auto-restart after a crash, the programs Bsys and Checkdsk are run to verify the consistency of the file system. If either of these programs detects errors which cannot be corrected automatically, the message "Tenex not available: Disk needs fixing" is broadcast to all terminals instead of the usual "Tenex in operation" message, and logins are prohibited from all terminals except the two in the Maxc room.

The following procedures require wheel or operator status and are intended principally for reference by system personnel. With some assistance from a user in the Maxc room, a system maintainer can perform these procedures from a home terminal. Only in extreme circumstances should non-system personnel attempt any of these procedures.

Errors detected by Bsys (which are usually reflected by some further errors detected by Checkdsk) indicate inconsistencies in the structure of user file directories. Fixing these requires a fairly intimate knowledge of the Tenex directory structure; this should be left to system personnel. Information about the structure of directories may be found in the Tenex Monitor Manual, section VII, pages 2-5. Other helpful information is available in the Bsys manual, pages 33-36. Copies of both these documents are kept in the Maxc room book case.

Checkdsk errors come in a number of guises. For each file with errors, data will be printed as in the following example:

```
<NEELY>MESSAGE.COPY;3      Filename
40050172166 MDA 0          } List of errors
140050172170 MDA 63        }
1 PTE                      } Error
2 MDA                      } summary
```

If there are many errors in a single file, Checkdsk will print out only the first few, followed by the summary. Study the output carefully.

First, note that "NOT IN BT" errors *have been corrected by Checkdsk,* so don't worry about them. If these were the only errors that occurred, Checkdsk wouldn't have complained and the system would have flown on.

The other kinds of errors reported by Checkdsk are more serious:

| | |
|---|---|
| MDA | Multiply-assigned disk address |
| IDA | Illegal disk address |
| PTE | Page table error |

MDA errors are the only ones that cause Tenex to prohibit users from logging in, since further file activity is likely to make the damage spread.

Note that you will have to use some *judgment* in discriminating between garbaged page tables and real MDA errors. A file with a garbaged page table will have an enormous error count (in the hundreds) with many categories of errors (IDA, MDA, PTE, etc). This is frequently caused by an untimely Tenex crash occurring between directory update and page table update during new file

creation, so that the page table for the file will not have been written on the disk yet and whatever was on that page before will be interpreted as a page table. This type of error may result in many *other* files getting bogus MDA errors because some of the entries in the garbaged page table look like valid disk addresses that happen already to be assigned.

For further confirmation that the problem is a garbaged (unwritten) page table, a QFD of the filename should reveal that it was created within a minute or two before the time of the last system crash. Such a file should be deleted using the following procedure:

```
@ENABLE password <cr>
!CONNECT <directory with bad file> <cr>
!DELETE <bad filename> <cr>
!EXPUNGE <cr>
!
```

This procedure causes the bad file to be expunged from the directory. A number of valid addresses possibly in use by other files may be deallocated, but don't worry about this. The system will generate a number of BUGCHKs for illegal disk addresses, but don't worry about this either. (Be sure DCHKSW is set to zero, however, to prevent the system from breakpointing on these errors). Run Checkdsk again after performing this surgery to make sure you did it right and that there is nothing else wrong. Checkdsk will reallocate pages incorrectly deallocated by the preceding procedure and will type out "NOT IN BT" for these.

```
!CONNECT SYSTEM <cr>
!CHECKDSK <cr>
REBUILD BIT TABLE? N
SCAN FOR DISK ADDRESSES? N
```

(This currently takes about 15 minutes).

After all files with garbaged page tables have been eliminated (if there were any), any further errors are considerably more serious, particularly MDA errors. MDA stands for multiply-allocated disk address, meaning that a particular page has somehow been assigned to more than one file. For each such error, Checkdsk has printed out the *second* file owning the page that it encountered in its scan of the file system; you do not yet know the name of the other owner of that page. Hence you should follow this procedure:

```
!CONNECT <directory containing file with MDA error> <cr>
!COPY <affected filename> GARBAGE <cr>
!DELETE <affected filename> <cr>
!EXPUNGE <cr>
!RENAME GARBAGE <affected filename> <cr>
```

Repeat this procedure for all affected files. You should be careful to type the <affected filename> in full, including version number, so you don't mistakenly fix up the wrong file.

Next, re-run Checkdsk as explained above. While running, Checkdsk will type out a number of NOT IN BT errors whose disk addresses correspond to the disk addresses in the original MDA error printouts; the filenames typed out will be those of the *other owners* of the pages that were

multiply assigned. It may not be obvious which owner of a page has the correct copy (a QFD of the filenames will include the write dates, which may give some indication; i.e. the file with the newer write date is more likely to have the correct data), but you have done the best that can be done by giving non-conflicting copies to everybody involved. Use SNDMSG to notify all users who have (potentially) lost files. Both the original MDA and the final NOT IN BT files are involved in the loss.

When you have pieced the filesystem back together to what you believe is a reasonable state, you should open the system to users by the following procedure:

```
!QUIT <cr>
./
FACTSW[ 500000,,0  400000,,0 <cr>
<control-P>
ABORT
.↑
!LOGOUT <cr>
LOGOUT JOB .....
<control-C>
```

After you type control-C, the auto-jobs should start logging in, and shortly thereafter "Tenex in operation" will be broadcast.

# 19. BSYS OPERATION

This section describes normal operating procedures for backup and archiving by means of BSYS. Further information on BSYS commands may be obtained from the document "BSYS - Tenex Backup System" by Smokey Wallace, a copy of which is kept in the Maxc room book case.

The procedures are identical for Maxc1 and Maxc2, with the exception that archiving is not presently supported on Maxc2. Separate sets of backup disk packs are used for the two systems.

One should be aware of what procedures can and cannot be performed simultaneously. Only one dump of any kind (incremental or full dump, archiving, or backup to tape) can be in progress at a time. However, it is ok to perform archive retrieval (or retrieval of files from backup) at the same time as a dump. Of course, since Maxc1 and Maxc2 are independent systems, there are no restrictions relating to simultaneous Bsys operations on the two systems.

## 19.1. Backup Procedures

A full dump of the file system is performed approximately once a month, and incremental dumps every weekday, onto disk packs mounted on auxiliary drives. At present, the backup packs consist of two sets of packs for full dumps and enough packs for incremental dumps to last about a month. Successive full dumps alternate between the two sets of full dump packs, and the incremental dump packs are then recycled until they are all rewritten with data more recent than the last full dump, at which point it is necessary to perform another full dump. This procedure ensures that (1) if the file system is wiped out, it will be possible to restore it to its state at the time of the last incremental dump, and (2) if a user loses a file and notices its loss within one month, it will be recoverable.

In addition to full and incremental dumps, a full backup to tapes is performed once every three months and the tapes stored in Building 34. This provides backup in case of a catastrophe such as a fire in the machine room.

BSYS is organized around the use of tapes as the backup medium. Disk packs have been interfaced to BSYS at a very low level, with the result that most of BSYS still thinks it is dealing with tapes. The backup packs are laid out in a manner that allows several logical "tapes" to be stored on each one. Every such "tape" has a "tape number" (just as with real tapes) whose format is "xxxyy", where xxx is the pack number and yy is the logical "tape" number (or "dump number") on that pack. Hence, "Parc tape number 21402" means the second dump on pack number 214.

When a full dump is performed, each pack contains a single logical "tape", since every pack except the last is completely filled by its respective portion of the full dump. An incremental dump, on the other hand, typically fills somewhat less than one pack. Hence it is possible to write incremental dumps for several successive days onto a single pack. When a pack becomes full, the last incremental dump will usually be split between the last logical "tape" on that pack and the first "tape" on the next pack. This is ok since each "tape" has its own directory and is completely self-contained. (However, the last incremental dump pack should not be allowed to overflow for obvious reasons.)

A record of usage of full and incremental dump packs is posted on the Maxc room bulletin board. It should be kept up to date.

### 19.2. Incremental Dumps

Every working day, an incremental dump should be performed, preferably during periods of light Maxc load (e.g. before 8 am or after 5 pm). The procedure is as follows.

1. Consult the "Incremental Dumps" chart to determine the pack currently being used for incremental dumps. Mount that pack on any free drive. Wait for it to be on-line before starting BSYS.

2. Login as someone with "wheel" or "operator" capabilities and execute the following procedure (user typein is underlined):

@ENABLE password <cr>
!CONNECT BSYS <cr>
!BSYS <cr>
TENEX BSYS 4.03 23-MAY-75
*INC<esc>REMENTAL DUMP (CHANGED FILES) <cr>
Dump since t&d? No <cr>
Entire file system? Yes <cr>

3. You will now be asked to "Enter backup device (mtan:, dpkn:)". Here you should enter DPKn, where n is the unit number (drive A is unit 0, B is unit 1, ... H is unit 7). Then you will be asked for the pack number. When you supply this, BSYS will attempt a Tenex "Mount" of the pack. Assuming this is successful, one or more of the following things will happen:

a) "Pack has no home block. Do you want to initialize the home block?" This should happen only for a freshly-initialized pack, i.e. one that has not been used for backup before. Since all our backup packs have been used many times, you should not see this question.

b) "Home block type is xxxxx. Do you want to initialize the home block?" The mounted pack has been used for some other purpose and should probably not be used as a backup pack. (This is especially true if xxxxx is "TENEX"!)

c) "n dumps already written on this pack", followed by a list of the dates and times of the dumps, followed by "Do you want to overwrite these dumps?" If you are appending to the pack that was used in the previous day's incremental dump (i.e. the most recent date printed out is yesterday), you should answer "No". If you are recycling a pack from the previous month (i.e. the dates printed are on the order of a month ago), you should answer "Yes".

d) If the number of free pages on the pack is less than 2000 and you answered "No" to the question about overwriting existing dumps, BSYS will say "Insufficient room for more dumps" and will ask you to mount another pack (on the same drive). The reason for this check is to ensure that it will be possible to write the file <SYSTEM>DIRECTORY on the first logical "tape" of this dump.

4. BSYS will then ask "Listing to file:". We keep incremental dump listings on the BSYS directory in files named BSYS/mm-dd-yy.INC, where mm-dd-yy is today's date. So you should respond with, for example, "BSYS/7-25-75.INC<cr>".

5. Finally, BSYS will ask you to "Enter tape id:". You should respond with the same string as you gave for the listing filename, e.g. "BSYS/7-25-75.INC<cr>".

6. The dump should now proceed. BSYS prints a summary of files and pages dumped for each directory (more detailed information is sent to the listing file).

7. If the pack becomes full, BSYS will once again request you to "Enter backup device (dpkn:, mtan:)". You should dismount the pack, mount the next incremental dump pack, and repeat step 3 above. In this case, you should always answer "Yes" to the question "Do you want to overwrite these dumps?" since you are recycling an old incremental dump pack. Be sure to update the chart on the bulletin board.

8. When the incremental dump is finished, BSYS will type an asterisk. The last step of the daily backup procedure is to do an "Expunge" on all directories:

```
*QUIT <cr>
!DISCUSE <cr>
 SYSTEM TOTAL: 6679 PAGES LEFT, 129401 USED
!EXPFILES <cr>
[Long pause while expunges are done]
!DISCUSE <cr>
 SYSTEM TOTAL: 9475 PAGES LEFT, 126705 USED
!
```

9. Tear off the listing and put it in the filing cabinet. *(Fanfold to 8.5" x 11" for filing.)*

10. Occasionally you should delete obsolete incremental dump listing files, i.e., ones for dumps that have been overwritten. This is most conveniently done using the program DELBSYSINC (on the BSYS directory). It requests a date and deletes all files of the form BSYS/mm-dd-yy.INC in which mm-dd-yy is earlier than the date specified.

### 19.3. Full Dumps

The full dump procedure takes 6 to 8 hours depending on load, so it is usually run during the day (unless you plan to be at Parc all night anyway).

1. Consult the "Full Dumps" chart and select the least recently used set of full dump packs. Be sure to update the chart. Mount the first pack on any free drive. Wait for it to be on-line before starting BSYS.

2. Login as someone with "wheel" or "operator" capabilities and execute the following procedure (user typein is underlined):

```
@ENABLE password <cr>
!CONNECT BSYS <cr>
!BSYS <cr>
TENEX BSYS 4.03 23-MAY-75
*FULL<esc> DUMP (ENTIRE FILE SYSTEM) <cr>
```

3. You will now be asked to "Enter backup device (mtan:, dpkn:)". Here you should enter DPKn, where n is the unit number (drive A is unit 0, B is unit 1, ... H is unit 7). Then you will be asked for the pack number. When you supply this, BSYS will attempt a Tenex "Mount" of the pack. Assuming this is successful, one or more of the following things will happen:

a) "Pack has no home block. Do you want to initialize the home block?" This should happen only for a freshly-initialized pack, i.e. one that has not been used for backup before. Since all our backup packs have been used many times, you should not see this question.

b) "Home block type is xxxxx. Do you want to initialize the home block?" The mounted pack has been used for some other purpose and should probably not be used as a backup pack. (This is especially true if xxxxx is "TENEX"!)

c) "n dumps already written on this pack", followed by a list of the dates and times of the dumps, followed by "Do you want to overwrite these dumps?" The dates printed out should be aproximately two months old. Since you are re-using a full dump pack, you should answer "Yes" to this question.

4. BSYS will then ask "Listing to file:". We keep full dump listings on the BSYS directory in files named BSYS/mm-dd-yy.FULL, where mm-dd-yy is today's date. So you should respond with, for example, "BSYS/7-25-75.FULL<cr>".

5. Finally, BSYS will ask you to "Enter tape id:". You should respond with the same string as you gave for the listing filename, e.g. "BSYS/7-25-75.FULL<cr>".

6. The dump should now proceed. BSYS prints a summary of files and pages dumped for each directory (more detailed information is sent to the listing file).

7. When the pack becomes full, BSYS will once again request you to "Enter backup device (dpkn:, mtan:)". You should dismount the pack, mount the next full dump pack, and repeat step 3 above.

8. When the full dump is finished, BSYS will type an asterisk. The last step of the monthly backup procedure is to do an "Expunge" on all directories:

```
*QUIT <cr>
!DISCUSE <cr>
 SYSTEM TOTAL: 6679 PAGES LEFT, 129401 USED
!EXPFILES <cr>
[Long pause while expunges are done]
!DISCUSE <cr>
 SYSTEM TOTAL: 9475 PAGES LEFT, 126705 USED
!
```

9. Tear off the listing and put it in the filing cabinet. *(Fanfold to 8.5" x 11" for filing.)*

10. Delete the listing file for the full dump that has just been overwritten.

### 19.4. Full Backup to Tape

The procedure for backup to tape, performed quarterly, is identical to the full dump procedure with the following exceptions:

1. The first command issued to BSYS should be "BACKUP" rather than "FULL".

2. The answer to "Enter backup device (mtan:, dpkn:)" should be MTAn, where n is the tape unit on which the backup tape is mounted.

3. The listing file should be BSYS/mm-dd-yy.BACKUP. After completion of the dump, this file should be printed in a small font (Gacha6R90) and then deleted.

### 19.5. Continuing Interrupted Dumps

BSYS records its state at the beginning of each logical "tape". If Maxc crashes in the middle of a dump, this information may be used to restart the dump at the appropriate point. The procedure is as follows:

1. Do *not* change packs or tapes, even if the dump began on a pack or tape different from the one currently mounted.

2. Start BSYS and issue the "Continue Dump" command. Your responses to the questions "Enter backup device" and the ones that follow should be the same as the answers you *most recently* gave to the same questions.

3. BSYS will print "Restarting at user xxxxx" and resume the dump (incremental, full, or backup).

### 19.6. Restoring Files from Backup

Individual files may be restored from backup packs by the following procedure.

1. Determine where the file was written, by poking around in the listing files (.FULL and .INC). When you have located in the listing file the name of the file to be restored, go up to the top of the page to find the "Parc tape number", which will be in the form xxxyy, where xxx is the pack number and yy the dump number as previously explained.

2. Mount the pack so determined on an auxiliary drive.

3. Go through the following dialogue (xxx and yy are the numbers determined above):

```
@ENABLE password <cr>
!BSYS <cr>
TENEX-BSYS 4.03 25-MAY-75
*RES<esc>TORE FILES (FROM TAPE) <cr>
```

Enter backup device (mtan:, dpkn:) DPKn <cr>
Mount pack number: xxx <cr>
Pack xxx mounted
n dumps already written on this pack
[Listing of dump numbers and dates]
nnn free pages
Enter dump # yy <cr>
Listing to file: TTY: <cr>
Mounting tape directory

4. BSYS will now prompt you with "Restore DPKn:", to which you should respond with the full filename to be restored (including directory). Editing and recognition may be used and "*" may appear in any field; however, BSYS's filename logic is not as good as Tenex's, and you should be careful to specify a complete filename (including version) before hitting carriage return.

5. After restoring the specified file(s), BSYS will again prompt with "Restore DPKn:", and further files in the same dump on the same pack may be restored. When you have no more files to restore, hit Delete.

### 19.7. Restoring the Entire File System

This is the procedure to be followed if the file system has been wiped out and it is necessary to restore it from backup.

1. Load Tenex using Micro-Exec, go into EDDT, set DBUGSW to 2, and start the system at SYSLOD. To the question "Do you really want to clobber the disc by re-initializing?" answer "Y". After some amount of churning and various error messages ("No SYSJOB", "No EXEC", etc.) you will end up in the Mini-Exec.

2. A copy of BSYS is kept on Save Area 4, so to load and start it:

.Load program from area 4.
.Start.
TENEX-BSYS 4.03 25-MAY-75
*

3. Consult the summary from the most recent incremental dump to determine the logical "tape" on which that dump started. This is in the form xxxyy, where xxx is the pack number and yy the dump number on that pack. Mount the pack. Note that *this might not be the current incremental dump pack.* (If there have been *no* incremental dumps since the most recent full dump, then just mount the *first* pack of that full dump.)

4. Go through the following dialogue:

*RES<esc>TORE FILES (FROM TAPE) <cr>
Enter backup device (mtan:, dpkn:) DPKn <cr>
Mount pack number: xxx <cr>

Pack xxx mounted
n dumps already written on this pack
[Listing of dump numbers and dates]
nnn free pages
Enter dump # yy <cr>
Listing to file: NIL: <cr>
Mounting tape directory
Restore DPKn: <*>*.*;* <cr>
Create users? Yes <cr>
Creating users...
Bypass restore checks? No <cr>

5. BSYS first obtains a copy of the <SYSTEM>DIRECTORY that was written in this dump. This directory is used to control what files get restored from this and all other backup packs, and is also used to restore the correct file attributes. This is why it is important that the first dump restored be the first logical "tape" of the most recent incremental dump.

6. BSYS now restores all appropriate files from the dump, then asks "More tapes?" to which you should answer "Y". You should now specify the previous dump on the same pack, or the last dump on the previous pack. You must restore *all the dumps* on each of the most recent full dump packs and on each of the incremental dump packs more recent than the last full dump. It is *essential* that the incremental dumps be restored in reverse chronological order, and that all incremental dumps be restored before any full dump.

7. When all files have been restored, answer "N" to "More tapes?"; then give the "Quit" command to exit BSYS and the Mini-Exec "Halt Tenex" command to stop the system. Now reload and start Tenex in the normal fashion.

BSYS also has a "RELOAD DIRECTORIES" command that may be used to restore the entire contents of one or more user directories from backup, but less than the entire system. You should start by reloading the first pack of the most recent incremental dump, just as for the full restore.

### 19.8. Archive Procedures

When users request that files be archived, the files are simply marked for archiving. Twice a week (generally on Monday and Thursday), BSYS is run to archive these files onto two different tapes and then delete the files from disk (if appropriate). At any given moment, there are two "current" archive tapes. Files from successive weeks are appended onto these tapes until they become full. The tapes then have their write rings removed and are kept "forever", and new tapes are used for further requests.

The effect of archiving a file twice is obtained by simply running the archive system twice, once with the first "current" tape mounted and once with the second. Generally, both tapes will end up containing the same information. However, occasionally a user will make a new request between the first archive run and the second. In this case, the file gets archived for the first time on the second tape and for the second time on the first tape next week. Hence the archive tapes may not be simply copies of each other.

When a user requests that an archived file be retrieved, the retrieval request is recorded in two places. File <SYSTEM>RETRIEVE-REQUESTS.TXT is a human-readable text file containing all relevant information (filename, tape numbers, requesting user, etc.) This file is deleted once a week. File <SYSTEM>RETRIEVE-REQUESTS.BINARY contains the same information in a form readable by BSYS. Every working day, BSYS is run to process accumulated retrieval requests and restore archived files from tape to disk.

### 19.9. Organization of the Archive Tapes

The archive tapes are consecutively numbered starting at 700.[1] First and second archive tapes are paired. Each tape is labelled with an identifier such as "BSYS/7-21-75.ARC/1" or "BSYS/7-21-75.ARC/2", where the date specified is the starting date for that tape, and the trailing digit distinguishes the first archive tape from the second. When an archive tape is full, the date on which it became full is also written on the label, and the write ring is removed. (Actually, we usually do this before the tape has become completely full so as to avoid having archive runs split across two tapes. However, no harm arises if this occurs.)

The tape is also marked with the tape unit on which it was written. When possible, files should be appended to a tape using the same drive on which the tape was first written, and first and second archive tapes should be written on different drives.

### 19.10. Archiving Files to Tape

Twice a week *(preferably Monday and Thursday)*, the following procedure should be followed to write files onto the archive tapes. User typein is underlined.

1. If the newest "first archive" tape (marked ".ARC/1") is not full (i.e. has no ending date), make sure there is a write ring in it and mount it on the drive noted on the label. If it is full, mount the first unused tape on unit 2 (if possible).

2. Login as yourself (assuming you are a wheel or operator; otherwise login as SYSTEM) and go through this dialogue:

```
@ENABLE password <cr>
!BSYS <cr>
TENEX BSYS 4.03 23-MAY-75
*ARC<esc>HIVE (USER FILES) <cr>
Archival period (days) = 999999 <cr>
Listing only? No <cr>
Listing to file: LPT: <cr>
```

---

[1] Tape number 999 was used on 6-28-79. At this point old tapes were scrounged up and the sequential numbering system went to pot. Tapes numbered 671 through 687 cover the period 6-28-79 to 8-28-79. Tapes numbered 039 through 540 cover the period 9-2-79 to 6-26-80. At this point new tapes were purchased and put into use. They are numbered consecutively from 1001 through 1098.

Entire file system? Yes <cr>
List to users? Yes <cr>
Preface with special message? No <cr>
Enter backup device (mtan:, dpkn:) MTAn <cr>
Density (n=800, p=1600): P

3. BSYS will now ask "New tape?". If you have mounted a new tape, answer "Yes"; BSYS will then ask you to supply the tape number. Otherwise, answer "No"; BSYS will ask you to verify that there is a write ring in the mounted tape.

4. BSYS will now space to the end of the used portion of the tape (if any) and start writing new files.

5. If the tape becomes full, BSYS will rewind it and again ask "Enter backup device (mtan:, dpkn:)". You should label the tape as full, remove its write ring, and put it away. Then mount the first unused tape (*NOT the "second" archive tape*) on the same drive, and go through this procedure:

Enter backup device (mtan:, dpkn:) MTAn <cr>
Density (n=800, p=1600): P
New tape? Yes <cr>
Really?? Yes <cr>
Parc tape number: nnnn <cr>

Be sure to label the new tape properly.

6. When the archive run is finished, BSYS will report the number of files and pages archived. Dismount the archive tape and put it away.

7. Repeat the entire procedure (steps 1-6) to write the "second" archive tape, using the tape marked ".ARC/2" if you append to an existing tape, and using tape unit 3 if possible. If the second archive tape becomes full and you get to step 5, be careful to use *another* new tape to continue archiving on. That is, don't append to the new archive tape written on during the first archive.

8. If either archive tape is almost full (less than 1/2 inch of unused tape thickness on the reel), mark both tapes as full and remove their write rings before putting them away.

9. Go to the Clover room, collect the two listings, and store them in the tape cabinet.

Normally, we archive only those files specifically marked by users for archiving. However, it is occasionally necessary to perform "forced" archiving of all files not referenced within the last n days (n is generally 90). The procedure for forced archiving is identical to the normal archiving procedure with the following exceptions:

1. The reply to "Archival period (days) =" should be 90 (or whatever interval is being used) rather than 999999.

2. The reply to "Preface with special message?" should be "Yes". BSYS will then give control to

SNDMSG, to which you should type an appropriate explanatory message, such as "This archive includes all files not referenced in the past 90 days. New users will find useful information pertaining to how the archive system works by logging in on MAXC and using either the SEE or COPY commands to read the file <DOC>Archive-System.Doc.", followed by control-Z.

### 19.11. Retrieving Files from Tape

The following procedure should be executed once every working day to retrieve files from the archive tapes.

1. Login as yourself (assuming you are a wheel or operator; otherwise login as SYSTEM) and go through this dialogue:

> @ENABLE password <cr>
> !BSYS <cr>
> TENEX BSYS 4.03 23-MAY-75
> *RET<esc>RIEVE (ARCHIVED FILES) <cr>
> Process retrieval requests for all users? Yes <cr>

2. BSYS now sorts through all pending retrieval requests and tells you which tapes to mount:

> Mount Parc tape number nnn
> Enter backup device (mtan:, dpkn:) MTAn <cr>
> Density (n = 800, p = 1600) P
> [Following question asked only the first time]
> Listing to file: TTY: <cr>

3. BSYS automatically restores all requested files from the mounted tapes, and automatically notifies users via SNDMSG. When BSYS is done, it prints an asterisk.

If anything goes wrong during the retrieval process, it is possible to retrieve files from the second archive tape. You may first have to issue an Interrogate command for the files that were not retrieved from the first tape. Then run BSYS Retrieve in the normal way, but when it asks you to mount the first archive tape, mount the second one instead. BSYS will tell you that you mounted the wrong tape, but will go ahead anyway, if you tell it to. (*Second archive tapes not found in tape cabinets are stored in boxes in the Maxc machine room behind Maxc1 disk drives.*)

## 20. LOADING THE NOVA DISK

*Maxc1 only.* This procedure need be followed only if the "Maxc Nova" disk has been wiped out and must be reloaded from scratch.

A. Get the current "10SYS" tape from the tape rack and mount it on unit 0.[1] Note that the tape unit above and to the left of the Nova should be selected as unit 0. Set the Nova console switches to 100022. Make sure the tape drive is in remote and at the load point.

B. Push "Reset" followed by "Program Load" on the Nova front panel. The Infoton should print out:

FULL(0) or PARTIAL(1)?

and you should type "0" which will clear the Nova disk directory, read-in a fresh copy of Nova DOS from magtape unit 0, and rewind the tape.

C. Load files from the 10SYS tape by typing the following:

LOAD/A/V MT0:n; INSTALL MAXCSYS.SY <cr>

where n is the tape file containing all system files (usually 2; consult the label on the tape jacket).

D. Now set the Nova's console switches to 100040. Subsequently, pressing "Reset" followed by "Program Load" will cause DOS to be re-booted from the disk, leaving the file system intact.

E. If at some later time the Nova should fail to boot from disk (e.g., because the MAXCSYS.SY file has been deleted or clobbered), it is possible to boot from tape. Mount the 10SYS tape on unit 0, set the console switches to 100022, hit "Reset", then "Program Load". When the Infoton prints out:

FULL(0) or PARTIAL(1)?

type "1", which reads in DOS from tape but does not affect files on disk. You should now restore DOS to the disk, as follows:

XFER MT0:1 MAXCSYS.SY
CHATR MAXCSYS.SY SPW; INSTALL MAXCSYS.SY

---

[1]It is customary to hang the tape jacket from the clip mounted on the door of the tape unit, and for novices to remove the write ring from tapes.

## 21. CONTENTS OF THE NOVA/ALTO DISK

*Maxc1 only:* The following is a fairly complete list of the files regularly kept on the Nova disk and on file 2 of the 10SYS tape.

DOS system files and standard Nova subsystems

| | |
|---|---|
| MAXCSYS.SY | DOS boot image |
| TBOOT.SV | Tape boot program |
| EDIT.SV | Editor |
| ASM.SV | Nova assembler |
| RLDR.SV | Loader |
| GEARS.SV | Ears printing program |
| NEDP.SV | Ethernet diagnostic program |
| FTP.SV, .BB | Ethernet file transfer program |

Maxc-related files

| | |
|---|---|
| POWER.SV | Turns Maxc power on and off |
| MICRO.SV | Microcode assembler |
| OMICRO.SV | Micro overlay |
| EMICRO | ? (needed by Micro) |
| MIDAS.SV | Microcode loader/debugger |
| SYS.MB | PDP-10 emulator microcode |
| PATCHES.MB | Patches for SYS.MB |
| TENLOAD | Midas commands to load microcode |
| TENCSUM | Midas commands to display recomputed checksums for newly-assembled microcode |
| MEXECGO | Midas commands to load microcode, start Micro-Exec |
| TENGO | Midas commands to load microcode, start Tenex |
| NVIO.SV | Nova I/O control program |
| DMPLD.SV | Maxc memory dump/load program |
| TMEM.SV | Maxc memory diagnostic (runs on Nova) |
| PER.SV | Analysis program for TMEM output |
| TR.SV | Tests microprocessor registers and memories from the Nova |

Microprocessor diagnostics and command files

| | |
|---|---|
| DGBASIC.MB | DGBASIC |
| DGP.MB | DGP |
| DGALU.MB | DGALU |
| DGIMH.MB | DGIMH |
| DGIML.MB | DGIML |
| DGRL.MB | DGRL |
| DGREG.MB | DGREG |
| DGI.MB | DGI |
| DGM.MB | DGM |
| DGMR.MB | DGMR |
| DBEG.MB | |

Additionally, there are usually "old" (obsolete) versions of some of the above files (preceded by the letter "O") and "new" (experimental) versions (preceded by the letter "N").

*Maxc2 only:* The following files are ordinarily kept on the Maxc2 Alto's disk, in addition to standard Alto subsystems.

Maxc-related files

| | |
|---|---|
| MICRO.RUN | Microcode assembler |
| MIDAS.RUN | Microcode loader/debugger |
| MIDAS.* | Various auxiliary files needed by Midas |
| SYS2.MB | PDP-10 emulator microcode |
| TENLOAD.MIDAS | Midas commands to load microcode |
| INIT.MIDAS | Midas commands to load and dump newly-assembled microcode |
| MEXECGO.MIDAS | Midas commands to load microcode, start Micro-Exec |
| TENGO.MIDAS | Midas commands to load microcode, start Tenex |
| ALTIO.RUN | Nova I/O control program |
| TM.RUN | Maxc memory diagnostic (runs on Alto) |
| MEMBASH.RUN | Maxc memory contention exercisor |
| SMIDIAG.RUN | Alto System Maintenance Interface diagnostic |

Microprocessor diagnostics and command files

| | |
|---|---|
| DGBASIC.MB | DGBASIC.MIDAS |
| DGP.MB | DGP.MIDAS |
| DGALU.MB | DGALU.MIDAS |
| DGIMH.MB | DGIMH.MIDAS |
| DGIML.MB | DGIML.MIDAS |
| DGRL.MB | DGRL.MIDAS |
| DGI.MB | DGI.MIDAS |
| DGM.MB | DGM.MIDAS |
| DGMR.MB | DGMR.MIDAS |
| DGREG.MB | DGREG.MIDAS |
| DBEG.MB | |

The current microprocessor diagnostic sources are ordinarily kept on the Maxc2 Alto disk as well.

# 22. SOFTWARE MAINTENANCE PROCEDURES

This section outlines maintenance procedures for selected pieces of software that are in fact still maintained at all. The primary intent is to document the location of the sources and to identify the current maintainers.

## 22.1. Midas

*Maxc2 only.* The current sources for Midas are kept on the "Maxc2 Midas" Alto disk (maintained by Fiala). Reasonably current sources are also kept on [IVY]<MAXC] or [MAXC1]<MAXC> directories: MSOURCES.DM along with MIDAS.RUN, LOADER.MB, and DEBUG.MB. The various Midas command files are kept on DGSOURCES.DM along with the sources for the microdiagnostics.

Before initializing Midas on a new disk, be sure that you have the following files ready:

| | |
|---|---|
| Midas.Programs | (see below) |
| Gacha10.al | The font Midas uses |
| Midas.Midas | The initialization command file |

Midas.Programs contains a list of file names separated by blanks, commas, or carriage-returns. The names must be typed in UPPER-CASE. This list serves two purposes. First, file pointers are built for all of the names to speed up OpenFile. Next, the list of names for the "Run-Program" command menu is built. If the file name contains no extension, then hint FP's will be built for both name.MB and name.MIDAS and name will be put in the "Run-Program" menu. (However, the hint FP's are not built unless the file exists, and the file name will not be put in the "Run-Program" menu unless name.MIDAS exists.) If the file name contains an extension, then it will be put in the quick OpenFile table, but won't appear in the "Run-Program" menu.

Midas creates and uses the following files:

| | | |
|---|---|---|
| Midas.State | ~40 pages | Built for quick init by Midas/I |
| Midas.Resume | ~40 pages | Built before calling AltIO for resume |
| Midas.FixUps | 2 pages | Built when external fixups occur in .MB files being loaded (Current microcode never uses this.) |
| Midas.Errors | 2 pages | Written when "Compare" fails |
| Midas.SymTab | 100 pages | Written when symbol table overflows core buffers |

Altogether this is about 200 disk pages. It is desirable to do this with a pretty clean disk, so that the files wind up contiguous on the disk and near to each other.

## 22.2. NVIO

*Maxc1 only.* NVIO is maintained by Taft using a Nova disk labelled "NVIO". Backup copies of the source files are kept in the <NVIO> directory on Maxc1.

The NVIO source files all have the extension .NS, while command files for assembling and loading these sources have extensions .AS and .CF. Instructions for building a new version of NVIO are included as comments at the beginning of the source file C.NS.

### 22.3. AltIO

*Maxc2 only.* AltIO is maintained by Taft using an Alto disk labelled "AltIO/Maxc2". Backup copies of the source and command files are kept in a single dump-format file <MAXC>ALTIO.DM on Ivy.

AltIO may be compiled and loaded by executing the command files CompileAltIO.cm and LoadAltIO.cm respectively. AltIO makes use of a number of other Alto software packages available from the <Alto> directory. These packages are listed in LoadAltIO.cm.

### 22.4. TM, MemBash, SMIDiag, Alto Microcode

*Maxc2 only.* These pieces of software are kept as dump-format files in the <Maxc> directory on Ivy. The files are TM.dm, MemBash.dm, SMIDiag.dm, and MaxcAltoCode23.dm. They are compiled and loaded by means of command files whose names follow the conventions used for AltIO.

### 22.5. Tenex and Diagnostic Microcode

Sources for Maxc1 and Maxc2 Tenex microcode are maintained on the "Maxc1/2 Tenex Microcode" Alto disk by Fiala. Sources for Maxc1 and Maxc2 microdiagnostics are maintained on the "Maxc1/2 Microdiagnostics" Alto disk by Fiala.

These are backed up by Alto dump files stored on the IVY <MAXC> directory. Common sources are used for both Maxc1 and Maxc2 with conditional assemblies used for parts of the microcode that are different on the two systems.

Cross reference listings of Tenex microcode sources are obtained by FTPing the sources to the Maxc1 ERF directory and doing RUNFIL CRREF.RUNFIL, which uses MCROSS to produce the cross reference listing. The same procedure is used for microdiagnostic sources except that the DCRREF.RUNFIL command file is used.

### 22.6. Tenex

The Tenex systems for both Maxc1 and Maxc2 are generated from a common set of sources maintained on Maxc1 in the <134> directory (for Tenex version 1.34). Differences between the two systems are dealt with by conditional assembly.

The RUNFIL scripts LOADMAXC1.RUN and LOADMAXC2.RUN may be used to assemble and load new versions of Tenex. Most of the source files are separately assembled for each system, yielding .REL files with extensions .1RL and .2RL. However, some of the assemblies do not depend on any system parameters, so Maxc1 and Maxc2 share common .REL files for these. When changes are made to any system parameters in PARAMS.MAC or PROLOG.MAC, all the .1RL and .2RL files should be deleted to force reassembly of them.

The result of assembling and loading a new Tenex is a file called MAXC1.SAV or MAXC2.SAV, plus some auxiliary files whose extensions are .MAXC1 or .MAXC2. The bug string file and loader map are also printed on Ears; these should be placed next to the log book in the Maxc room.

After loading a new Tenex for Maxc1, all that is necessary is to run Micro-Exec and install MAXC1.SAV on an unused save area (e.g., 19) for stand-alone checkout. The "Read.Tenex.From.-File" and "Write.Tenex.To.Area" commands should be used for this purpose.

After constructing a new Tenex for Maxc2, one should transfer the various required files to Maxc2 by means of PUPFTP. These files are <134>MAXC1.SAV, <SYSTEM>BUGTABLE.MAXC2 and <SYSTEM>MONSYMS.MAXC2. These files should be transferred to the same directories and preserving the version numbers. They may then be deleted on Maxc1. Then the file MAXC2.SAV should be installed on a Maxc2 save area as explained above.

## 23. LOCAL MEMORY CHIP CHARTS

The most common cause of hardware problems in Maxc is failure of the storage chips used in the processor's local memories (LM, RM, IM, SM, DM, and MAP). Once the addresses and bit number(s) of the failures have been determined (by means of the micro-diagnostics DGRL or DGIML/DGIMH), the chips may be located by means of the information in this section.

The chips used in LM and RM are either Intel 3101A or TI 74S289, which are 16-word by 4-bit memories. They are located on the three ALUA boards, each one of which provides a 12-bit slice of the 36-bit Maxc ALU. In the following table, the top three rows show the card slot number as a function of the machine and bit number, and the main matrix below it yields the chip position on that card as a function of the memory name and address within the memory.

| | **Maxc1** Card Slot | Bits | | | **Maxc2** Card Slot | Bits | | |
|---|---|---|---|---|---|---|---|---|
| | 2/25 | 0-3 | 4-7 | 8-11 | 2/23 | 0-3 | 4-7 | 8-11 |
| Memory & | 2/21 | 12-15 | 16-19 | 20-23 | 2/21 | 12-15 | 16-19 | 20-23 |
| Address | 2/17 | 24-27 | 28-31 | 32-35 | 2/19 | 24-27 | 28-31 | 32-35 |
| LM 0-17 | | 25 | 26 | 27 | | 44 | 24 | 64 |
| LM 20-37 | | 37 | 38 | 39 | | 45 | 25 | 65 |
| RM 0-17 | | 1 | 2 | 3 | | 55 | 35 | 75 |
| RM 20-37 | | 13 | 14 | 15 | | 54 | 34 | 74 |

**Figures 1, 2 & 3** in the **Appendix**, prepared by Ron Weaver, show the storage chip layouts for the old and new bipolar memory cards used in the microprocessor. Old bipolar cards are used in all positions in Maxc1 but in only the MAP slot in Maxc2; new bipolar cards are used in all other positions in Maxc2. Note that the bits on the old bipolar cards are arranged differently for Maxc1 and Maxc2.

If it is necessary to replace an entire card, one should be aware that not all the bipolar cards are directly interchangeable. In particular, old bipolar cards in four of the IM slots have had pullup resistors removed and cannot be interchanged with cards in the other slots. New bipolar cards used in IM slots have different pullup resistor chips (680 ohms) than are used in SM/DM slots (330 ohms); these must be changed if cards are substituted.

# 24. CREATING AND DESTROYING MAXC ACCOUNTS

## 24.1. Obtaining a Maxc Account

Maxc directories are requested via a directory-request form obtainable from Haychan Sargent. The following forms are available:

Files-only directory application **(Figure 7, *Appendix*)**
CSL, SSL, SDD application (Palo Alto employees only) **(Figure 8, *Appendix*)**
Xerox non-Palo Alto or non-Xerox application **(Figure 9, *Appendix*)**

The top part of a directory application is filled out by the applicant and his supervisor. The form then migrates to Hal Murray (SDD), Larry Masinter or Ted Strollo (SSL), or Ron Weaver (other) who does the following:

Verifies that the directory name is not in conflict with an existing directory and that the password is reasonably unguessable;

Approves the directory request and fills in the directory parameters on the form;

For ordinary cases, the account administrators fill out the form, sign it, create the directory, and send the form to R. Weaver, who in turn signs the form, create the account, and passes the form on to Haychan Sargent, who is in charge of Maxc account files. A non-standard case is passed to E. Fiala for approval before the directory is created. Directory creation of requests is normally carried out by Ron Weaver (<RWEAVER>), who may also create directories for the other account administrators.

The various steps in creating a Maxc directory are discussed below. After the directory is created, the application form goes back to Haychan Sargent who files the form and passes the new directory information to Kathi Anderson for updating message distribution list files and Lynn Harrington for updating the phone lists.

For PARC personnel, the Maxc directory application may also be used as an application for an IFS directory on IVY. The requestor checks the "MAXC Account" and/or "IVY Account" box to request a Maxc and/or Ivy directory. Note that only one application is filed for a directory, regardless of whether that directory exists on only Maxc, only Ivy, or both.

## 24.2. General Information About Maxc Directories

*Passwords*

Initial passwords are required to be unpronounceable and relatively unguessable. However, there is nothing to prevent the user from changing his password later, so we can't guarantee this. We do not permit passwords for "regular" directories (as opposed to message-only directories) to be known to more than one person; this guards the system against unauthorized access.

Upper and lower case characters are equivalent for passwords, so it is not necessary to be careful about case while entering passwords.

*User groups and Directory groups*

Tenex file and directory protection distinguishes between three classes of access:

self (someone logged into or connected to the directory);
same group (someone logged into a directory in the same group);
others.

Putting a user's login directory into a User Group allows him to access other directories in that group in the *same group* mode.

Putting a directory into a Directory Group allows other users in that group access in the *same group* mode.

Directories are usually setup so that new files created in that directory will permit total owner access, read-only same-group access, and no access to others. Most new directories are assigned to the Computer-research user group and directory group. There are many exceptions.

A list of the current groups is supposed to be kept in <ACCOUNTS>GROUP-LIST.NUMBERS and was as follows on 4 October 1979:

| | | | |
|---|---|---|---|
| 0 | Computer-research | 12 | Graphics |
| 1 | System | 13 | Pogos |
| 2 | Mesa | 14 | Nova |
| 3 | Secretaries | 15 | PARC |
| 4 | Lisp | 16 | XMS-users |
| 5 | Understander | 17 | User-Sciences |
| 6 | Bliss | 18 | ITG |
| 7 | Inter-Network | 19 | Mesa-users |
| 8 | Customer Programming | 20 | Reyes |
| 9 | University MPC | 21 | SSL Group Leaders and Secretaries |
| 10 | Alto | 22 | JDS |
| 11 | Parcpub | 23 | University Grant |

Most Xerox user directories are put in the Computer-research directory and in no other groups.

*File Protection*

The file protection assigned a directory is the default protection of new files created in that directory. The file protection is an 18-bit number divided into three 6-bit groups. This number breaks down into three fields, identical in format, each of which can be regarded as containing two octal digits or six binary bits. The first 6-bits define owner protection, second same-group protection, and third general-public protection. The interpretation of the bits in a group is as follows:

B0 Read contents of file
B1 Write onto file
B2 Execute program stored in file
B3 Append to file
B4 Access per page table
B5 --

Setting a bit to 1 permits the action; setting it 0 denies the action. Taking these six bits as a two digit octal number, some common values are: 77, which permits full access; 52, which protects a file from modification but permits other functions; and 00, which denies everything. [Refer to **Figure 6, *Appendix*** for correct values corresponding to boxes on Files-only account request form.]

*Directory Protection*

In addition to specifying the access allowed to file contents, Tenex allows directory contents to be protected similarly. The directory protection word is composed of three 6-bit fields, one field each for "self", "group", and "others" similar to the file protection word. The bits have meaning as follows:

B0 If off, completely prevents use of the directory in any way
B1 Files may be opened subject to file protection
B2 Owner functions permitted (including CONNECT) without password
B3 Files may be added to the directory
B4 --
B5 --

Directory protection can only be changed by someone with WHEEL status. Some common values of Directory protection are: 776060, which permits the owner to do everything, other users to read-write files according to file protection; 777760 permits the owner and same group to do everything, others to read-write according to protection (typically used for files-only directories); 777700 permits owner and same group everything, general public nothing (typically used for files-only directories that one doesn't want non-group people to see); 777060 permits owner everything, same group connect access (typically used for files-only directories in the Computer-research group, where you want everyone to be able to connect but not to be able inadvertently to clobber files while not connected). [Refer to **Figure 6, *Appendix*** for correct values corresponding to boxes on Files-only account request form.]

*Disk Limit*

Presently, message-only directories are assigned a disk limit of 50 which is enforced.

Regular directories are assigned a disk limit that varies according to the organization: CSL, 1000 pages; SSL, 750 pages; SDD, ASD, and other, 250 pages. This varies when special needs are present. Disk limits for regular directories are not "enforced", which means that file storage in these directories can exceed allocation without difficulties (except an annoying "over allocation" message from Tenex) until total free storage in the system falls below 2000 disk pages. When free storage is below this, attempts to write new files into directories over allocation are denied.

*Project Group*

The project group assigned a directory classify it for the accounting software, which summarizes CPU utilization, file storage use, etc. according to these groups.

The current project groups are CSL, CSL-Summer, CSL-Consultants, CSL-Files-only, Defunct, SSL-unknown, SSL, SSL-RI-PT-Contract-VS, SSL-Consultants, SSL-Files-Only, University-MPC, Lisp-Files-Only, ICL, SDD, Mesa, GSL, JDS, OSL, Administration-not-on-phone-list, Administration, SDD/LA, ADL, PARC, PD/LA, ED/LA, Xerox, Non-Xerox, Computer-Research, Tenex, ASD/PA, ASD/ES, Webster, EOS/Pasadena, Old-NonPARC and Versatec. Directories are attached to these projects by editing the <ACCOUNTS>PROJECTLIST file.

The PROJECTLIST file is used by the MAXC-ACCT subsystem discussed later. It is important that this file be updated correctly, but Tenex will run correctly even when this file is invalid or incomplete.

The format of a group in the PROJECTLIST file is as follows:

```
CSL:
AIS, ALISP, ..., WINOGRAD, YEARY;
```

i.e., the group name followed by a ":", the list of directory names in alphabetical order separated by "," and terminated by ";".

*Files-only Directories*

Files-only directories are created when needed, usually to hold files needed by a number of different users working on a common project. Tenex does not allow login to files-only directories. Each files-only directory is represented by a line in the <SYSTEM>ARCHIVE-FILES-ONLY.TXT file of the form:

```
OWNER,FILEDIRECTORY
```

where OWNER is the login directory name of the person responsible for FILEDIRECTORY. <SYSTEM>ARCHIVE-FILES-ONLY.TXT is needed by the archive system to direct messages about archive traffic to OWNER. By convention, the entries in ARCHIVE-FILES-ONLY.TXT are in alphabetical order by owner, and where there are several files-only directories for a single owner, these are also in alphabetical order.

<SYSTEM>ARCHIVE-FILES-ONLY.TXT is also used by the HOGS program, discussed below, to associate disk usage of files-only directories with the login account for the person responsible for the storage.

*Login Directories*

Each login directory is represented by a line in the file <ACCOUNTS>UACHK.TXT. The standard entry in this file is as follows:

```
ACCOUNTANT:1*,BACKGROUND
```

This line controls the pie slice scheduling groups accessible to the directory. The "1*" means that the default login pie slice is the numerically named account number 1; the ",BACKGROUND" means that after login, the user can change his pie slice to the low priority BACKGROUND slice.

Nearly all directories are represented by a UACHK.TXT entry in the above format. However, a few directories for people who maintain the hardware or do other special stuff, may be allowed to access some other slices. For example:

SYSTEM:1*,BACKGROUND,SERVICE,MAINT,ARPANET,220100

This allows the <SYSTEM> directory to change its scheduling slice to BACKGROUND, SERVICE, MAINT, ARPANET, or 220100.

The UACHK.TXT file is compiled into a big accounting matrix which is stored on <SYSTEM>UACHK.FILE during the procedure for creating or destroying directories.

*Message User*

Message-only users have very limited access to subsystems and have enforced disk limits. The subsystems accessible to message-only users are specified in <SYSTEM>SUBSYSTEMS.directory-name, if it exists, or in <SYSTEM>SUBSYSTEMS.DEFAULT, if no special file exists for the directory. It is required that subsystem names in these files be in alphabetical order. <SYSTEM>SUBSYSTEMS.DEFAULT presently contains the following:

<PUP>EFTP.SAV
<SUBSYS>ARCVER.SAV
<SUBSYS>DELVER.SAV
<SUBSYS>MSG.SAV
<SUBSYS>PLZFIX.SAV
<SUBSYS>PRESS.SAV
<SUBSYS>READMAIL.SAV
<SUBSYS>SEE.SAV
<SUBSYS>SNDMSG.SAV
<SUBSYS>XMS.SAV

If a special <SYSTEM>SUBSYSTEMS.directoryname file is created for a message-only directory, it will generally contain all of these files plus the extra ones that are added for the directory.

### 24.3. The E^CCREATE and E^CPRINT Commands

Directories are created and destroyed using the Tenex Executive's E^CCREATE command (i.e., <control-E>CREATE), which has subcommands that allow properties of the directory to be specified. The various subcommands for E^CCREATE are discussed here, and then examples where directories are actually created and destroyed are given in the following sections. The current properties of any directory can be observed by using the E^CPRINT command. User encrypted password is printed out as two octal numbers by using the "verbose" form of the ↑E Print commnad (that is, "↑EPrint dirName Verbose"). This is useful in comparing Maxc1 and Maxc2

passwords, or transferring a password from one system to the other, without knowing the clear-text version of that password. You must have enabled your WHEEL or OPERATOR status to execute CREATE or PRINT.

CREATE and its subcommands:

!E$^{C}$CREATE username [NEW] (password) password,<cr>
[SUPERPASSWORD] GUESS<cr>

The default parameters of a new directory are as follows:

DISK limit = 750 pages (not enforced)
DEFAULT file protection = 775200
PROTECTION of directory = 776060
USER group number = 0 (Computer-Research)
DIRECTORY group number = 0 (Computer-Research)
NUMBER of directory = next unused number
no special capabilities
has a mailbox (i.e., has a MESSAGE.TXT file)

If the directory is not supposed to be in the Computer-Research user group or directory group, you will have to remove it from this group using the NOT DIRECTORY and/or NOT USER subcommands below.

| | |
|---|---|
| !!ABORT | = C$^{C}$ |
| !!NUMBER dirnumber | for setting directory number. When creating a new directory on Maxc1, the directory number is defaulted to the next unused number. However, on Maxc2, this subcommand is required to ensure identical directory numbers on both Maxc1 and Maxc2. [1] |
| !!FILES | makes a files-only directory |
| !!MESSAGE | makes a message-only directory |
| !!MAINTENANCE | a capability--subset of WHEEL/OPERATOR capabilities |
| !!OPERATOR | a capability |
| !!WHEEL | a capability |

---

[1] When you run out of directory numbers (highest possible = 1777), you can reset the last-used directory number, as follows:

```
@enable
!quit
./
1[ 0
call mapdir<esc>x
lstdno[ 10
<control-P>
.↑
!
```

This resets the last-used directory number to 10. Be sure to do this on both Maxc1 and Maxc2. Subsequently, new directories you create will re-use directory numbers of directories that once existed but have been deleted.

| | |
|---|---|
| !!DEFAULT<esc> (file) PROTECTION 6-char octal number | sets the default file protection for directory |
| !!DIRECTORY groupnumber | argument is one of the directory group numbers given earlier--repeat this subcommand for multiple directory groups |
| !!SECONDARY groupnumber | argument is one of the directory group numbers given earlier--repeat this subcommand for multiple directory groups |
| !!USER groupnumber | argument is one of the directory group numbers given earlier--repeat this command for multiple directory groups |
| !!DISK limitinpages | sets disk limit |
| !!ENFORCE | causes disk limit to be enforced |
| !!PROTECTION<esc> (of directory) dirprotec | sets directory protection |
| !!LIST | prints the status of the directory as E^CPRINT would do if the CREATE command were terminated now |
| !!NO MAILBOX[1] | Prevents MESSAGE.TXT file from being created. This will be issued on either Maxc1 or Maxc2 for regular and message-only accounts. In other words, if the primary system for a user is on Maxc1, his mailbox will be on Maxc1; if the primary system is Maxc2, then his mailbox will be on Maxc2. When Laurel has been developed further, there will probably be a number of user directories that do not have mailboxes on either system. The NO MAILBOX command is not required for files-only directories because the FILES subcommand automatically does this. |
| !!NOT DIRECTORY groupnum | removes the directory from a directory group |
| !!NOT SECONDARY groupnum | removes the directory from a secondary directory group |
| !!NOT ENFORCE | don't enforce disk limit |
| !!NOT FILES | not a files-only directory |
| !!NOT MAINTENANCE | turns off the MAINTENANCE capability |
| !!NOT MESSAGE | turns off the message-only restriction |
| !!NOT OPERATOR | turns off the OPERATOR capability |
| !!NOT USER groupnum | removes the directory from a user group |
| !!NOT WHEEL | turns off the WHEEL capability |
| !!PASSWORD | for changing password |
| !!ENCRYPTED (password) | Sets password, using encrypted form (two octal numbers) as input. |
| !!KILL | kills or deletes the directory |

---

[1]Only has effect when creating a new directory. Has no effect when used on an established directory. MSGFIX must be used on established directories to create or delete mailboxes.

Unused subcommands:

| | |
|---|---|
| !!ABSOLUTE | |
| !!ALPHANUMERIC | |
| !!MODE | |
| !!NAME | for changing directory names--unimplemented |
| !!REPEAT | obsolete |
| !!RETENTION | unimplemented |
| !!CONFIDENTIAL | a capability |
| !!NETWIZARD | a capability |
| !!PRIVILEGES octalnum | setting capabilities in octal |
| !!SPECIAL | |

### 24.4. Creating a Maxc Directory

You have to be a WHEEL or OPERATOR to create or destroy a Maxc directory, change the password, or do other kinds of directory maintenance.

1. Enable yourself and connect to the ACCOUNTS directory on Maxc1. Then do a WHO or LD and make sure that none of the other users who diddle with the accounting are at that time also modifying the accounting files. They would be connected to the ACCOUNTS or SYSTEM directory, if they were doing this.

2. For login and message-only directories (as opposed to files-only directories) edit UACHK.TXT to contain the new directory name. For files-only directories edit <SYSTEM>ARCHIVE-FILES-ONLY.TXT to contain the new directory name. For all directories edit <ACCOUNTS>-PROJECTLIST to contain the directory in the appropriate places. If you are creating a number of directories, you can do the edits for all of the directories at once, which saves time.

For some message-only directories, additional subsystems will be specified; these will require the creation of a <SYSTEM>SUBSYSTEMS.username file which should include the contents of <SYSTEM>SUBSYSTEMS.DEFAULT plus the additional subsystem names. We follow the convention of preserving alphabetical ordering of the names in each section of these files.

3. E^CCREATE the directory on Maxc1 using one of the following examples as a prototype:

*Message-only directory:*

| | |
|---|---|
| !E^CCREATE username [NEW] password,<cr> (Don't omit the ",") | |
| [SUPERPASSWORD] GUESS<cr> | |
| !!MESSAGE<cr> | Makes message-only directory |
| !!DISK 50<cr> | Sets the disk limit to 50 pages |
| !!ENFORCE<cr> | Enforce the disk limit |
| !!LIST<cr> | List the properties of the directory and make sure that all the parameters are setup as desired |
| !!<cr> | Execute the CREATE with properties as setup |
| [CONFIRM]<cr> | |

*Files-only account:*

| | |
|---|---|
| !E$^{C}$CREATE username [NEW] password,<cr> [SUPERPASSWORD] GUESS<cr> | (Don't omit the ",") |
| !!FILES<cr> | Makes files-only directory |
| !!LIST<cr> | List the properties of the directory and make sure that all the parameters are setup as desired |
| !!<cr> [CONFIRM]<cr> | Execute the CREATE with properties as setup |

*Regular directory:*

| | |
|---|---|
| !E$^{C}$CREATE username [NEW] password,<cr> [SUPERPASSWORD] GUESS<cr> | (Don't omit the ",") |
| !DISK 1000<cr> | Sets the disk limit to 1000 pages |
| !!LIST<cr> | List the properties of the directory and make sure that all the parameters are setup as desired |
| !!<cr> [CONFIRM]<cr> | Execute the CREATE with properties as setup |

4. After creating the directory you should print the parameters of the directory and write down the directory number, which you will need shortly:

| | |
|---|---|
| !E$^{C}$PRINT username<cr> [SUPERPASSWORD] GUESS<cr> | |
| DISK LIMIT 1000 | |
| WHEEL | |
| ALPHANUMERIC ACCOUNTS | |
| DIRECTORY NUMBER 15 | *Write down this number for later* |
| DEFAULT FILE PROTECTION 500000775200 | |
| DIRECTORY PROTECTION 500000776060 | |
| USER GROUPS: 0, 1 | |
| DIRECTORY GROUPS: 0, 1 | |

5. If you are creating a number of directories at once, you should carry out a E$^{C}$CREATE sequence for each of them at this time.

6. Next, you have to reinitialize the accounts on Maxc1 as shown below, and then copy the resulting stuff to Maxc2 where a similar but slightly different sequence is carried out:

| | |
|---|---|
| !RUNFIL UACHK.RUNFIL<cr> | Reads UACHK.TXT and creates UACHK.FILE on the connected directory (you are still connected to <ACCOUNTS>) [This step is necessary for message-only and regular directories but is not needed for files-only directories.] |

--Check for any error messages before continuing--

!E^CINITIALIZE ACCOUNTS<cr> Renames UACHK.FILE onto the <SYSTEM> directory and installs it as the current accounting matrix [This step is necessary for message-only and regular directories but is not needed for files-only directories.]

!PUPFTP MAXC2<cr>
*CONN ACCOUNTS<cr>
*PRESERVE VERSION<cr>
*STORE PROJECTLIST<cr><cr>
<User password incorrect
LOGIN (user) YOURDIRECTORY yourpassword<cr>
*STORE <SYSTEM>UACHK.FILE<cr><cr>

copy <SYSTEM>UACHK.FILE on Maxc1 to <ACCOUNTS>UACHK.FILE on Maxc2. [This step is necessary for message-only and regular directories but is not needed for files-only directories.]

*CONNECT SYSTEM
*STORE <SYSTEM>ARCHIVE-FILES-ONLY.TXT<cr><cr>

You only have to do this copy if you modified <SYSTEM>ARCHIVE-FILES-ONLY.TXT. You also have to copy <SYSTEM>SUBSYSTEMS.username if you added or modified any of these files.

*QUIT<cr>
!DET<cr>

--Chat to Maxc2--

@ENABLE yourpassword<cr>
!CONN ACCOUNTS<cr>
!E^CCREATE username [NEW] password,<cr> (Don't omit the ",")
[SUPERPASSWORD] GUESS<cr>
!!same subcommands as on Maxc1
!!NO MAILBOX<cr>

You only create a mailbox, i.e., a MESSAGE.TXT file, on one of the two Maxc systems. Normally the mailbox is created on Maxc1 and you give the NO MAILBOX subcommand on Maxc2. However, for a primary user of Maxc2, the mailbox is put on Maxc2 and you would issue NO MAILBOX when you CREATEd the account on Maxc1.

!!NUMBER directory-number (from E^CPRINT above)
!!<cr>
[CONFIRM]<cr>

If you are creating a number of directories, then you go ahead and create the others at this time, remembering to add the NO MAILBOX and DIRECTORY dirnumber subcommands in addition to the stuff you did on Maxc1. Then when you have created all of them, you finish up with the following:

| | |
|---|---|
| !E^CINITIALIZE ACCOUNTS<cr> | Initializes the accounts, renaming <ACCOUNTS>UACHK.FILE to <SYSTEM>UACHK.FILE [This step is necessary for message and regular directories but not needed for files-only directories.] |

Note that if you have a number of Maxc directories to create at once, you can insert all the names into the UACHK.TXT, PROJECTLIST, and ARCHIVE-FILES-ONLY during the same editting session. Then you can E^CCREATE all the directories. Then you do the RUNFIL, INITIALIZE ACCOUNTS, and PUPFTP of the files just once. Then you Chat to Maxc2, CREATE all the directories there, and finally do the INITIALIZE ACCOUNTS just once on Maxc2.

### 24.5. Editing the Grapevine Data Base

The Grapevine data base must be edited to contain the correct directory name and password for all new Maxc accounts. It must also be edited to delete the directory name for all Maxc accounts that are destroyed. To do this you should have version 6T15 or later of Laurel so you can run program Maintain, the Grapevine Registration Server Maintenance Program.

Run Laurel and bug "Run" in the lower command menue. Type Maintain<cr> and control will pass to the Grapevine Registration Server Maintenance Program. A typical session follows:

**Creating a Mailbox:**

Grapevine Registration Server Maintenance Program
Version unknown

GV: Login [Confirm] Yes
Your Name Please: *Name.pa<cr>*
Your Passoerd: *password<cr>* . . . Locating registration server . . . 3#14#51 . . . ok
GV: Create Individual : *Name.pa<cr>* with password: *password<cr>* . . . done
- -> Add Mailbox at server: *Maxc<cr>* for individual: *Name.pa<cr>* . . . done
GV: Type Entry for R-Name: *Name.pa<cr>* . . . done, type=individual
Connect-site:
Forwarding: null
Mailbox-sites: maxc
GV: Quit [Confirem] Yes
*End of Message*

**Deleting a Mailbox:**

GV: Delete Individual: *Name.pa<cr>* [Confirm] Yes . . . done
- -> Remove All memberships in registry: pa for R-Name: *Name.pa<cr>*

**Mail Forwarding:**

GV: Add Mailbox at server: *Maxc<cr>* for individual: *NewName.pa<cr>* . . . 3#14#51 . . . done
GV: Remove Mailbox at server: *Maxc<cr>* from individual: *OldName.pa<cr>* . . . 3#14#51 . . . done
GV: Add Forwarding to: *NewName.pa<cr>* for individual: *OldName.pa<cr>* . . . 3#14#51 . . . done
GV:

**Type "?" to see available commands.**

GV: ? Commands are:
Add Forwarding, Add Friend, Add List of Members, Add Mailbox, Add Member, Add Owner, Create Group, Create Individual, Delete Group, Delete Individual, Initialize New Name, Login, Modify All Occurrences, Quit, Remove All Memberships, Remove Friend, Remove Forwarding, Remove Mailbox, Remove Member, Remove Owner, Set Connect-site, Set Decimal Password, Set Password, Set Remark, Set Server, Type All Groups, Type Details, Type Entry, Type Members, Verify All Groups, Verify Group, Verify Name
GV:

**Creating List of Mailboxes:**

GV: Type Entry for R-Name: Individuals.*pa* . . . done, type = group
Remarks: Individuals.*pa*
Members: AHenderson.pa, ABell.pa, - - -
- - - etc.
GV: Type Entry for R-Name: Individuals.es . . . done, type = group
Remarks: Individuals.es
Members: Abagaz.es, Abe.es, - - -
- - - etc.
GV: Quit [Confirm] Yes

Now PUT the Maintain.Typescript in some file (i.e. Individuals.GV) and use Bravo to hardcopy.

### 24.6. Changing the Password and Other Modifications to Directories

The E^C CREATE command is also used to modify parameters for an existing directory. When used in this way, directory parameters are initially defaulted to the current parameters of the directory, so only those items being changed need be typed as subcommands.

Note: It is illegal to change the directory number or the name of the directory in this way. To do this it is necessary to destroy and recreate the directory and copy files manually.

When using CREATE in this way you omit typing the password, so that you don't have to be told the password of the directory in order to change its parameters. For example:

```
!E^CCREATE olddirectoryname,<cr>   (Don't omit the ",")
 [SUPERPASSWORD] GUESS<cr>
!!NOT USER 0<cr>                   Remove from Computer-research user group
!!USER 1<cr>                       Add to System group
!!NOT DIRECTORY 0<cr>
!!DIRECTORY 1<cr>
!!PASSWORD xglot1<cr>              Change the password
!!<cr>
 [CONFIRM]<cr>
```

### 24.7. Destroying a Maxc Account

Maxc directories are destroyed after either an explicit request from a Maxc directory owner (rare), after the sign out procedure that occurs when someone leaves PARC, or when we find out by some other means that a directory is dead. Ron Weaver is notified when someone leaves PARC, and he either destroys the Maxc directory (most desirable) or converts it to files-only (less desirable but hard to avoid if there are too many still-useful files archived out of the directory). Directories for users outside PARC may become dead, but we have no convenient method of automatic discovery.

Prior to destruction, the directory owner should delete unnecessary files and rename useful ones into the directory of a coworker. Users are sometimes sloppy about cleaning up their directories, so the person who is going to kill the directory should, as a safety measure, delete obsolete versions of files and archive current versions as follows:

```
!CONNECT olddirectory<cr>
!DELVER<cr>
!ARCHIVE FILE *.*;*<cr>
---prints out what is going to be archived---
!DELETE *.*;*<cr>
---deletes archived files, but not those about to be archived---
```

*Note:* If files exist on Maxc2, get someone to clean up the directory or move them to Maxc1 for archiving.

Then wait for the weekly or twice-weekly archiving run to take place. After that, all information about files will be contained in the archive directory, and you can continue with the procedures discussed below. If the directory does not have an archive directory, this safety procedure cannot be followed.

The person who is about to destroy the directory should list *one copy of the archive directory before destruction*, as follows:

```
!CONNECT olddirectory
!INTERROGATE *.*;*,
!!EVERYTHING<cr>
!!LPT<cr>
!!<cr>
```

This copy is stapled to the original directory application and filed in the CSL dead directory file.

When a directory is destroyed, any files remaining in it will be deleted. It is less automatic to retrieve archived files belonging to a deleted directory. We have avoided reusing directory numbers for defunct directories so that, if we have to, we can then recreate the directory to retrieve files from the archive. We expect to continue this policy until we run out of directory numbers (probably in 1980). Retaining a listing of the archive directory is a new practice, so directories destroyed before 19 April 1978 can readily be accessed only by recreating the directory.

The procedure for destroying a directory involves retrieving the original directory request form from our files (Haychan Sargent), marking it as destroyed, and refiling. For PARC directories, the form should also be considered for destroying IVY IFS directories for the person who is departing, if that is appropriate.

To destroy a Maxc directory, proceed as follows:

1. Login to Maxc1, enable yourself, and connect to the <ACCOUNTS> directory. Do a LD or WHO to verify that none of the other account maintainers are possibly modifying the account information at that time.

2. Edit the accounting files to remove the directory being destroyed. PROJECTLIST has to be editted for all directories, UACHK.TXT for login directories, and <SYSTEM>ARCHIVE-FILES-ONLY.TXT for files-only directories. <SYSTEM>SUBSYSTEMS.old-directory (if it exists) has to be archived for message-only directories.

3. After removing the appropriate entries from these files, destroy the directory with the following dialog:

```
!E^CCREATE olddirectory,<cr>  (Don't omit the ",")
 [SUPERPASSWORD] GUESS<cr>
!!KILL<cr>
!!<cr>
 [CONFIRM]<cr>
```

If you have a number of directories to destroy, you can delete all of them at this time, and then do the stuff below just once for all of them.

```
!RUNFIL <cr> COMMANDS FROM : UACHK.RUNFIL<cr>
!E^CINITIALIZE ACCOUNTS<cr>
!PUPFTP MAXC2<cr>
*CONNECT ACCOUNTS<cr>
*PRESERVE VERSION<cr>
*STORE PROJECTLIST<cr><cr>
 <User-Password incorrect
 LOGIN (user) yourself yourpassword<cr>
*STORE <SYSTEM>UACHK.FILE<cr><cr>
*CONNECT SYSTEM
*STORE <SYSTEM>ARCHIVE-FILES-ONLY.TXT<cr><cr>
*Q<cr>
!DET<cr>
```

Chat to Maxc2

```
@ENABLE yourpassword<cr>
!CONN ACCOUNTS<cr>
!E^CCREATE old-directory,<cr>
 [SUPERPASSWORD] GUESS<cr>
!!KILL<cr>
!!<cr>
 [CONFIRM]<cr>
```

If you have a number of directories to destroy, you can delete all of them at this time, and then do

the stuff below just once for all of them. If you deleted any SUBSYSTEMS.dirname files on Maxc1, then delete them on Maxc2 as well.

```
!E^CINITIALIZE ACCOUNTS<cr>
!
```

For each directory destroyed, a history of the directory is left in <ACCOUNTS>DELETED-USER-PARAMETERS.directoryname(directorynumber), and if it has any archived files, the archive directory is moved to <ACCOUNTS>DELETED-ARCHIVE-DIRECTORY.directoryname(directory-number). Also, don't forget to edit the Grapevine Data Base (Section 24.5.)

### 24.8. Operations on MESSAGE.TXT Files

It is sometimes necessary to create a mailbox in a directory that doesn't have one, to delete a mailbox, or to move a mailbox from Maxc1 to Maxc2, or vice versa. Since MESSAGE.TXT files have a funny "permanent" attribute that can't be set or cleared by any Exec commands, you have to create and destroy mailboxes using the MSGFIX subsystem, which requires that you enable yourself. It has commands for creating and deleting mailboxes (i.e., MESSAGE.TXT files). When a mailbox is deleted, its contents are copies into a file called OLDMESSAGE.TXT. The command sequences to MSGFIX are as follows:

```
!MSGFIX
*Create MESSAGE.TXT for directory: DIRNAME <cr>
*Q
!
```

to create a mailbox, or:

```
!MSGFIX
*Delete MESSAGE.TXT in directory: DIRNAME <cr>
*Q
!
```

to delete a mailbox.

It sometimes happens that a MESSAGE.TXT file has an excessive number of pages because the file has not been truncated for some reason. This can be remedied by using the following safe and simple procedure:

```
!CONN dirname<cr>
!RENAME MESSAGE.TXT FOO<cr>
!APPEND FOO MESSAGE.TXT<cr>
!DELETE FOO<cr>
```

In the case of a race occurring, the order of the old and new messages gets reversed, but no messages are lost. Note that MESSAGE.TXT files acquire extra pages only when manipulated by MSG, so only MSG users (never Laurel users) should suffer this problem.

When a MESSAGE.TXT file suffers a disk error, the recovery procedure is:

```
!CONN dirname<cr>
!RENAME MESSAGE.TXT BADMESSAGE.TXT<cr>
```

```
!TECO<cr>
*;Y<esc>
Input File: BADMESSAGE.TXT<cr>
(unexpected daya error interrupt)
*;U<esc>
Output File: BADMESSAGE.TXT<cr>
*
```

Then tell the user that BADMESSAGE.TXT may be retrieved for examination with Bravo or whatever. The bad data should NOT be put back into MESSAGE.TXT, as that might cause the internal structure of MESSAGE.TXT to be ruined.

### 24.9. Reinstantiating a Destroyed Directory

Sometimes a person whose Maxc directory has been killed needs to have that directory recreated. This could be done by creating a brand new directory, but it is usually more convenient to recreate the directory from the information that was saved when it was previously destroyed.

The way to do this is to retrieve <ACCOUNTS>DELETED-USER-PARAMETERS.DIRNAME-(DIRNUMBER) and <ACCOUNTS>DELETED-ARCHIVE-DIRECTORY.DIRNAME(DIRNUM-BER) from archival storage (i.e., they will usually have been archived by the time that you want to recreate the directory); in some cases the DELETED-ARCHIVE-DIRECTORY file won't exist. Then recreate the directory from these two files, as follows:

```
!E^CRECREATE DIRNAME<cr>
[SUPERPASSWORD] GUESS<cr>
```

This recreation procedure only works if the directory number has not been reassigned to some new directory in the interim. If it has been reassigned then you must treat the creation as if it were a new directory. Some special things must be done to the Archive-Directory if one existed before. (See Ed Taft for details.)

This reinstates the directory on Maxc1. Then add DIRNAME to PROJECTLIST and either UACHK.TXT or <SYSTEM>ARCHIVE-FILES-ONLY.TXT just as though a brand new directory were being created, and carry out the other procedures associated with creating a new directory on Maxc1. Then copy <ACCOUNTS>DELETED-USER-PARAMETERS.DIRNAME(DIRNUMBER) to Maxc2, RECREATE the directory on Maxc2, etc.

If the directory should have a MESSAGE.TXT file, then you should create the MESSAGE.TXT file with MSGFIX as discussed in the previous section.

### 24.10. Retrieving Archived Files for Defunct Directories

The easiest way to do this is to recreate the directory as discussed in the previous section, and then retrieve the files from archival storage in the normal way.

An alternative method of retrieving archived files from defunct directories is to find out on what tape the file is archived, and use the BSYS "Single File" command to copy it from the tape to a disk file in some existing directory. Unless the archive directory for the deleted directory was listed prior to destruction, the tape containing a particular archived file will generally be unknown. Directories destroyed after 19 April 1978 have a listing of the archive directory on file in the CSL dead directory file.

### 24.11. Printing Accounting Information

Running <ACCOUNTANT>USERAC.SAV will produce a printout of all the information on all the directories (requires enabled wheel status). The various options for this subsystem will be revealed when you type "H" as a subcommand.

The accounting printout periodically posted on the CSL bulletin board is obtained by connecting to the <ACCOUNTANT> directory and running <ACCOUNTANT>MAXC-ACCT, which renames the fact files from the <SYSTEM> directory into <ACCOUNTS> and churns through all the fact files over the time period selected.

RUNFIL <ACCOUNTS>HOGS.RUN will produce an output file named HOGS.DATE that shows the disk storage in use and the storage tied up by obsolete versions of files or files unreferenced in 90 days.