9100A/9105A Version 4.2 Software Release

Customer Information

The 4.2 software release of the 9100A is a maintenance update to the 4.0 release. The 4.2 release contains performance enhancements, fixes for reported bugs, and provides support for the 9100A-017 Vector Output I/O Module not present in 4.0. These notes provide a brief description of features added since 4.0, a set of instructions for installing version 4.2 software on a 9100A, a list of bugs fixed since the 4.0 software release, and a list of bugs known to exist in version 4.2.

P/N 877774
August 1989

No Text on This Page

1. Introduction

This document accompanies version 4.2 of the 9100A/9105A operating software. The document is divided into sections. Section 2 gives a brief description of the changes in version 4.2 of the system software. Section 3 describes the items you should have received with your 4.2 upgrade. Section 4 describes the procedure for updating your unit from version 3.0 or later software to version 4.2. Section 5 describes the bugs fixed between version 4.0 and 4.2. Section 6 describes the bugs known to remain in version 4.2. Section 7 consists of inserts for your TL/1 reference manual describing new and modified TL/1 functions, as well as corrected manual pages for some existing functions.

IMPORTANT NOTE

Because of new features added since version 4.0, the amount of memory available for user programs has decreased slightly. It is possible, but unlikely, that programs that ran under version 4.0 will not run under version 4.2 software unless you add memory to your 9100A/9105A.

It is expected that any program which ran under version 4.1 will also run under 4.2 without additional memory.

2. Features Added Since Version 4.0

2.1. Changes Between Version 4.0 and 4.1

The 4.1 software release was a maintenance update to the 4.0 release. The 4.1 release added performance enhancements, fixes for reported bugs, new TL/1 functions, and support for the 9100A-017 Vector Output I/O Module.

Changes to TL/1

- The **random** function returns a 32-bit numeric value from a psudo-random sequence. The function's optional argument is a seed value.

  ```
  r = random()
  ```

  Although the sequence of numbers generated by random has the appearance of randomness, no specific random property of these numbers is guaranteed. The sequences of numbers generated by this function may change in future software revisions.

- The **podinfo** function has new arguments that give more information about the current address space. The new options 'read', 'write', and 'run' return 1 if the indicated operation is permitted in the current space, and 0 otherwise.

  ```
  p = podinfo run      ! p = 1 if RUN UUT is permitted
  p = podinfo read     ! p = 1 if reads are permitted
  p = podinfo write    ! p = 1 if writes are permitted
  a = podinfo busaddr ! a = default bus test address
  ```

- The **getpod** function returns the name of the current pod database. Its one argument is 'podname'. There is a current pod (called 32BIT) even when no pod is plugged in. 9132A pod names always begin with an 'M', so getpod is useful to determine the type of pod as well as the name.

  ```
  n = getpod podname
  ```

  getpod may eventually return more information about a pod. Getpod must be called using keyword notation.

- The **cwd** function returns the current working directory (the user disk and directory from which program execution was begun) as a TL/1 string. If the program was started from a UUT, the string has the form "/userdiskname/uutname". If the program was started from a pod library, cwd returns the drive and pod name as a string of the form "/userdiskname/PODLIB/podname". If the program was started from the proglib, the cwd returns the string "/userdiskname/PROGLIB".

  ```
  declare string dirname

  dirname = cwd()
  ```

  This function permits the reliable construction of absolute path names on the current disk. Some TL/1 string manipulation will be required.

  Cwd returns the name of the directory from which execution began, not the directory containing the currently executing program. This is important for programs being debugged or executed in the podlib or proglib.

- The **filestat** function returns the read/write status of a text file as a TL/1 string of the form "xxx" if the named file exists, and the null string if the file does not exist. The first character position of the string contains 'r' if the file is readable and '-' otherwise. The second position contains 'w' if the file is writable and '-' otherwise. The third position contains '-'.

  Filestat may eventually report more information about the file, so do not rely on the length of the string to remain the same in the future. However, new information will be added to the end of the string so existing programs will continue to work. Filestat should only be called using keyword notation.

  ```
  declare string s

  s = filestat file "/hdr/test_status"
  ```

- The **loadblock** function loads a block of data into UUT memory. It operates like writeblock, but takes an additional numeric argument called 'offset'. Offset is interpreted as a two's-complement value added to the base address of each block in the data file to obtain the load address. If the file contains position-independent code, loadblock can relocate the code anywhere in UUT memory.

- The **readblock**, **writeblock**, and **loadblock** functions now understand Intel Hex format as well as Motorola S-record format.

- The **readword**, **writeword**, and **setword** functions permit reading and writing of I/O module pins in a manner similar to the IOMOD INPUT WORD, IOMOD OUTPUT WORD, and IOMOD SET WORD functions of the operator's interface.

  setword selects between one and forty pins of an I/O module to be grouped together so they may be read and written as a unit. Up to five groups may be created per I/O module.

  writeword writes a pin group. Each pin may be written high, low, or put into a high impedance state.

  readword reads the value of a pin group. The current values of the pins can be read, or the clocked level history from the last measurement can be read.

```
setword device "/mod1", word 1, as_pins "3 6 10 13"
setword device "/mod1", word 2, as_pins "1 2 4 5 8 9 11 12"
writeword device "/mod1", word 2, patt "11100100"
output = readword device "/mod1", word 1, mode "now" ! current pattern
if output <> "0111" then
  print "NAND gate test failed"
end if
```

- The **dbquery** function has two new options. 'dbquery inputs' returns a comma-separated list of a pin's related inputs. 'dbquery pintype' returns the pin's type as a string ("INP", "OUT", "BID", "PWR", "GND" or "UNU").

```
list = dbquery inputs "U2-3"          ! returns related inputs for U2-3
type = dbquery pintype "conn1-14"     ! returns pin type for conn1-14
```

- Improvements in memory management within TL/1 programs have made some programs more efficient in their use of memory. Thus some programs that ran out of memory under previous software versions can now run. Delays related to memory management have also been reduced.

- Certain TL/1 operations are much faster. Though it is difficult to characterize what operations are faster, the improvement is most noticeable in programs that do a lot of arithmetic or string operations. Loading time has also been reduced, but not dramatically.

- If the user disk is set to one of the floppy disks, /dr1 or /dr2, TL/1 also searches for programs on the hard disk's podlib and proglib if the program is not on the floppy disk.

Changes to the E-Disk

- The E-disk will now accept a UUT from more than one disk. This permits the 9105A to use UUTs larger than a floppy disk.

  If the UUT being loaded has the same name as the UUT in the E-disk, contents of the UUT will be added to the E-disk, rather than replacing them. For example, if a UUT named "TK80" was loaded from a floppy drive onto the E-disk, the floppy was ejected and another floppy disk inserted, and TK80 was loaded from the second floppy onto the E-disk, the contents of the first "TK80" UUT would be combined with the contents of the second "TK80" UUT. Thus, UUT's can be broken up over several floppies but loaded together onto the E-disk.

  If a TL/1 program, GFI database or test vector file is loaded onto the E-disk when a program, database or test vector file of the same name already exists on the E-disk, the newly-loaded file will over-write the previously-loaded file.

  If you want to load a UUT with the same name as the UUT that was previously loaded, and you do not want the two UUT's combined, first select the DELETE E-DISK menu option, then load the UUT.

- Vector files can be loaded into the E-disk. On the operator's display, the E-disk Load menu displays a choice of "WITH VECTORS" or "WITHOUT VECTORS".

  Test vectors are fairly large files; it is recommended that you load these only if they will be used heavily.

Vector Output I/O Module Support

- The following functions have been added to TL/1 to support the Vector Output I/O module.

clockfreq	drivepoll
edgeoutput	enableoutput
syncoutput	strobeoutclock
vectordrive	vectorload

These functions are described fully in the manual insert accompanying the Vector Output I/O Module. **These names are now reserved, and programmers should not name programs with these names.**

- The editor now handles vector data. This function is also explained fully in the user manual accompanying the Vector Output I/O Module.

2.2. Changes Between Version 4.1 and 4.2

The 4.2 software release is a maintenance update to the 4.0 and 4.1 releases. The 4.2 release contains additional performance enhancements and fixes for reported bugs.

- Customers occasionally reported that TL/1 programs running under version 4.1 of the system software would suddenly begin to run slowly for no obvious reason. Changes to the internal memory management software have eliminated this problem.

  As a byproduct, many programs now run significantly faster; in some cases nearly 100 per cent faster. Programs may load up to 50 per cent faster. Also, some programs which ran out of memory under 4.1 may now be able to run.

- TestROMfull stopped with an error if it attempted to diagnose a faulty ROM, and the address increment was not a power of two.

  Since the diagnostic routine must use address increments that are powers of two, the address increment is rounded down to the next smallest power of two for diagnosis, and the ending address is also adjusted.

  Although the ROM test will now work with any address increment, we recommend powers of two be used for the best and most understandable results.

- Several minor bugs in the Vector Output I/O module software were fixed. These bugs are listed in section 5.

3. Version 4.2 Upgrade Kit Contents

3.1. 9100A-903

This kit upgrades version 3.0 9100A or 9105A systems to version 4.2.

System Disk 1 -- Ver 4.2
System Disk 2 -- Ver 4.2
Manual Update Package
4.2 Release Notes
Options Binder
Master User Disk 1 -- Ver 4.1
Master User Disk 2 -- Ver 4.1

3.2. 9100A-903K

This kit upgrades version 4.0 or 4.1 9100A or 9105A systems to version 4.2.

System Disk 1 -- Ver 4.2
System Disk 2 -- Ver 4.2
4.2 Release Notes
Master User Disk 1 -- Ver 4.1
Master User Disk 2 -- Ver 4.1

3.3. 9100A-904

This kit upgrades version 3.0 9100A systems with programming to version 4.2.

System Disk 1 -- Ver 4.2
System Disk 2 -- Ver 4.2
Programmer's System Disk 1 -- Ver 4.2
Manual Update Package
4.2 Release Notes
Options Binder
TL/1 Reference Card
Master User Disk 1 -- Ver 4.1
Master User Disk 2 -- Ver 4.1
Demo Trainer Disk -- Ver 3.0

3.4. 9100A-904K

This kit upgrades version 4.0 or version 4.1 9100A systems with programming to version 4.2.

System Disk 1 -- Ver 4.2
System Disk 2 -- Ver 4.2
Programmer's System Disk 1 -- Ver 4.2
4.2 Release Notes
Master User Disk 1 -- Ver 4.1
Master User Disk 2 -- Ver 4.1
Demo Trainer Disk -- Ver 3.0

4. Installing New System Software

The procedure for installing new system software depends on whether the main frame is a 9100A or 9105A.

9100A Software Installation Procedure

The following procedure should be followed to install a new version of the system software on a 9100A which is running version 3.0 or later software.

IMPORTANT NOTE

If your 9100A is running Version 2.2 or earlier software, it must be upgraded to Version 3.0 before Version 4.2 can be installed. Failure to update from 2.2 to 3.0 before installing Version 4.2 will leave your 9100A in an unusable state.

IMPORTANT NOTE

Once the installation procedure is started, do not restart or power down your 9100A until all disks are copied. If you do, you run the risk of leaving your 9100A in an unusable state.

(1) Copy the new System Disks and Programmer's System Disk (if you have the Programmer's Option) to the hard disk by selecting the command:

MAIN: COPY DISK FROM DR1 TO HDR

on the front panel display.

When you load new system disks from the factory, the 9100A writes validation data on them. The validation data allows the software to be loaded. If you receive the message

PLEASE WRITE ENABLE SYSTEM DISK

remove the disk, push the write-enable tab on the disk to the closed (write enabled) position, and re-insert the disk. Once validated, the system disks cannot be loaded or copied on any other machine.

Write the serial number of the 9100A (located on the rear panel) on each of

the disks to avoid mixing them up with disks from other systems.

(2) Restart your 9100A by depressing the recessed RESTART button (on the right side panel), or by turning the power switch (on the back panel) off, then on. This operation loads the new system software into memory. The 9100A should boot up normally and display the new software version number.

(3) The original system disks should be stored in a safe place as backups.

(4) If your upgrade includes new Master User Disks, you may copy them to the 9100A hard disk by selecting the command:

MAIN: COPY DISK FROM DR1 TO HDR

on the front panel display.

IMPORTANT NOTE

Any programs that have been modified on your 9100A hard disk, and have the same name as those on the master user disks, will be overwritten.

9105A Software Installation Procedure

The following procedure should be followed to install a new version of the operating software on the 9105A.

(1) Restart your 9105A by depressing the recessed RESTART button (on the right side panel), or by turning the power switch (on the back panel) off, then on.

(2) Use the new System Disks to boot the 9105A.

The first time you load new system disks from the factory, the 9105A writes validation data on them. The validation data allows the software to be loaded. If you receive the message

PLEASE WRITE ENABLE SYSTEM DISK

remove the disk, push the write-enable tab on the disk to the closed (write enabled) position, and re-insert the disk. Once validated, the system disks cannot be loaded or copied on any other machine.

Write the serial number of the 9105A (located on the rear panel) on each of

the disks to avoid mixing them up with disks from other systems.

The 9105A should boot up normally and display the new software version number.

(3) Make a working copy of each of the system disks. The originals should be stored in a safe place as backups. To copy the disks, follow the procedure in "Duplicating a Disk Using a 9100A/9105A" in the Applications Manual.

5. Bug Fixes and Bug Notes

The following bugs reported in previous versions of the system software have been corrected in version 4.2. The bug descriptions and workarounds (if any) are furnished for reference only; they are not needed for programs running under version 4.2 or later.

Bugs Fixed Between Version 4.0 and 4.1

870716-07: **TL/1 CHECK unwisely clears syntactic/lexical error messages.**

DESCRIPTION

If a program on disk contains lexical or syntactic errors, error messages are added to the text as it is loaded. However, if you do a CHECK, the messages disappear.

880222-02: **Editor MARK/CUT bug in reflist and part.**

DESCRIPTION

Using MARK/CUT to delete lines from a reflist or part can cause the editor to lose track of where the cursor should be. The cursor may appear on the wrong line, or the screen may get garbaged as you use the cursor movement keys to move around (up-arrow, down-arrow, etc.). The editor may even lock up.

WORK AROUND

MARK/CUT small regions that do not cause the display to scroll. Also, when the following message appears:

'x lines deleted <PRESS RETURN>'

press RETURN to reposition the cursor.

881020-03: **Key sequences execute unintended previous function.**

DESCRIPTION

Operations like PROBE, that have more than five softkey choices work strangely when recorded as a sequence.

To repeat, press PROBE key, FREQ softkey, then ENTER. Press SEQ BEGIN ENTER. Press PROBE ARM ENTER. Press SEQ QUIT ENTER. Press SEQ PERFORM ENTER. It does a FREQ measurement.

881108-02: **Editor COPY file type PART to TEXT problem.**

DESCRIPTION

When a PART file is copied to a port, the arrows and special symbols in the PART file are converted to printable ASCII characters that appear similar. However, if a PART file is copied to a TEXT file, the same arrows and special symbols are not converted to printable ASCII characters.

Either conversion (PART to PORT or PART to TEXT) ought to produce the same result.

881109-01: **Display of UUT directory is not updated after UUT compile.**

DESCRIPTION

Edit a UUT and scroll the display window so that the DATABASE line of the directory is displayed. Now COMPILE the UUT. The newly-created database does not appear in the UUT directory until you SCROLL-BACKWARD and then SCROLL-FORWARD. The display is not being refreshed properly.

This problem also exists for UUT coverage report.

881117-01: **ROM data fault reports address range incorrectly**

DESCRIPTION

When testromfull is used in a TL/1 program, and the addrstep argument is defaulted, the test operates incorrectly. In word-wide address spaces, the test would stop with an error "Invalid address step". In byte-wide address spaces, fault messages would have an incorrect value for the 'upto' argument.

WORK AROUND

The 'addrstep' argument is supposed to default to the default address increment for the current space. In 3.0 or later releases, this can be simulated by explicitly including the addrstep argument, as in addrstep (podinfo addrinc). A valid constant value (usually 1 or 2) also works fine.

881122-02: **Use of gfi suggest or gfi accuse in GFI stimulus programs will not generate expected recommendations.**

DESCRIPTION

When GFI is run from the front panel, it:

(1) tests a pin
(2) examines the state of pins on the UUT to generate new probing recommendations
(3) updates the display, showing the IC, pin messages and the recommendation.

But if a stimulus program calls 'gfi suggest' or 'gfi accuse', then GFI does not generate new recommendations. This causes it to display: NO RECOMMENDATION, or to recommend probing the pin that was just tested.

890110-01: **Related pin lists on parts with pin names don't work.**

DESCRIPTION

If the pins on a part description have names (rather than pin numbers), then the related pin lists are ignored by GFI.

WORK AROUND

Use pin numbers rather than pin names.

890117-01: **Print followed by input does not flush the print buffer.**

DESCRIPTION

print followed by input does not flush the print buffer
open as output mode buffered
open as input mode unbuffered

WORK AROUND

open as input mode buffered

but this requires the ENTER key to terminate the input cmd.

890220-01: **Message for Built-in fault m_pod_rom1_cs is incorrect.**

DESCRIPTION

9132A Pod bustest occasionally reports: "ROM1 CS or OE is stuck invalid". This message comes from the TL/1 built-in fault m_pod_rom1_cs. In the situations where this occurs, the ROM's CS or OE pin is not stuck inVALID, but rather it is stuck inACTIVE. To say it is invalid (i.e. neither 1 nor 0) is incorrect and misleading to the operator. The message should read: "ROM1 CS or OE is stuck inactive".

890220-02: **Extremely large GFI databases cause problems in LEARN.**

DESCRIPTION

When a GFI LEARN ONE NODE was done in a certain large response file, learn inserted an extra line into the file. The extra (second) line contained unstable signature data as shown:

```
LD2-9   PROBE     0000   X        TRANS   0   <-- correct line LD2-9
PROBE     *   *    *   TRANS   0   <-- extra line
```

This bug only manifests itself when the GFI database is really large. (This one had over 49,000 reference designator pins.)

WORK AROUND

Delete added lines manually.

890222-03: **Erroneous "E-disk too small" message during E-disk allocation.**

DESCRIPTION

1) Load an E-disk successfully.

2) Go to the editor (this should delete the E-disk).

3) Quit the editor.

4) Try to load the e-disk with the same parameters you used the last time. An error message comes up saying the E-disk doesn't have enough memory allocated (even though it worked fine before).

WORK AROUND

Repeat the allocation one more time. The second time, it works.

890227-02: **Function calls with defaulted arguments sometimes fail mysteriously.**

DESCRIPTION

In certain programs, when a function was called with some of its arguments defaulted, the function returned an error message even though the default value was perfectly appropriate in context.

The programs affected all used floating point numbers. In detail, what happened was a function call whose n'th argument was floating, followed by a function call whose n'th argument was defaulted, and was a string or number. The functions setoffset() and arm() seemed to be particularly common, since their first argument, which defaults to "/probe" is often defaulted.

WORK AROUND

Things work fine if explicit arguments are provided.

890227-03: **Default GFI-RUN "REF" and "PIN" not erased when GFI is cleared.**

DESCRIPTION

When GFI is cleared, the default "REF" and "PIN" fields under the front panel GFI-RUN menus should also be cleared.

890309-01: **Errors in TL/1 on-line quick reference file.**

DESCRIPTION

The TL/1 On-line Quick Reference file (reached by pressing the HELP key while editing a program) has the following errors:

No description of 'loadblock' in the file

:readblock: The 'format' argument does not list the other possible value, i.e. "intel"

:runuutvirtual: The description in the comment is wrong; 'extaddr

0, addr $FFFFFFFE' should give a 64 bit virtual address of $00000000FFFFFFFE, not $0000000000001A2D2

:setspace: The description of 'space' says it uses the value returned by "getspace and setspace"; this actually should be "getspace and sysspace"

:str: and :val: The list of possible values for 'radix' should also include 2 and 8

:testbus: The 'addr' argument value is $FFFE in the example, but the comment says FFFF

:testromfull: The 'sig' argument value should be $AE38 (leading $ sign is required)

:writevirtual: The description in the comment is wrong; 'extaddr 0, addr $100' should give a 64 bit virtual address of $0000000000000100, not $0000000000001A2D2

The TL/1 on-line quick reference file also has the following minor problems:

:count: :level: :sig: :storepatt: and :writepin: The 'refpin' argument is not mentioned in the description

:connect: The example program line is split between two lines, which makes it appear that program lines can somehow be continued on another line

:declare: The description of "default value" and 'array' is not in the right place

:end: The 'end <program name>' syntax is described twice, on consecutive lines

:for: The 'step' argument is not mentioned here (but is mentioned under :loop:)

:function: The example should say something about "user defined" or "example", to make it clear that 'max' is a sample function, and not the description of a command called 'function' that returns the

larger of two values passed to it

:print: The description in the comment not quite correct; when 'on' is not specified, the output is to the operator's display only if no output channel was opened or if the display was the first output channel opened

:readvirtual: :runuutvirtual: and :writevirtual: The default value is not given for the 'extaddr' argument

:strobeclock: The default value for the 'device' argument ("/probe") is not given

WORK AROUND

When in doubt, go by the book (except when the manual is also wrong).

890320-01: **Bug in 68000 QWK_ROM program.**

DESCRIPTION

There is a bug in the 68000 QWK_ROM program.

WORK AROUND

In the program PODLIB/68000/QWK_ROM line # 97 should be:
    return (readspecial addr $F000300C)
instead of:
    return (read addr $F000300C)
otherwise an "Illegal Address" fault will occur.

In line # 99 the read command should be readspecial as well.

Bugs Fixed Between Version 4.1 and 4.2

890202-01: **Modified TL/1 program suddenly slows down.**

DESCRIPTION

When the AUTO_LEARN TL/1 program was modified in certain ways, it began to run very slow, for no obvious reason. After a considerable portion of the program had run, response increased to more typical levels.

WORK AROUND

These symptoms result from problems in TL/1's memory manager. The unpleasant symptoms can often be made to vanish by declaring a large global array. The memory consumed by the array encourages TL/1 to allocate a new block of memory from the system, which usually makes things better.

Start with 200 elements and keep making it larger until the performance returns to normal. The symptoms will often go away if you make the program any larger or smaller.

890310-01: **F6 STYLE softkey missing in PODLIB and POD directories.**

DESCRIPTION

The PODLIB and POD directories do not have the F6 STYLE softkey.

890310-02: **Illegal initial value type in TL/1 declaration is not detected.**

DESCRIPTION

If a variable is given a default value that is the wrong type, the program does not detect the error. For example;

```
declare
  string S = 0    ! incorrectly assigned a numeric default value to a string
  numeric N = ""  ! incorrectly assigned a string default value to a number
end declare
```

If the example above were in a TL/1 program, neither the line syntax check or the CHECK softkey run check routine would find the error. When the program is executed, no error message is given, either. In this case, the string variable is not assigned any value, and the numeric variable is set to 0. Less pleasant things can happen if a non-zero numeric value is assigned to the string.

WORK AROUND

Correct programs are not affected by this problem.

890322-01: **I/O Module not armed when External Start argument is 'at_arm' and sync is 'ext'.**

DESCRIPTION

When the External Start argument of an I/O Module is set to clocked level history hardware of the I/O Module are not started when an I/O module is armed.

WORK AROUND

Apply a free running clock to the External Start Line of the I/O Module, or make sure that a RISING edge is applied to the External Start Line.

890406-01: **Capture offset value not reset after calibration.**

DESCRIPTION

The calibration value for the capture sync mode is set to the same value of EXT calibration of an I/O Module. After EXT calibration of an I/O Module, both only the EXT offset value and CAPTURE offset value need to be reset. The CAPTURE offset was not being reset to 0.

WORK AROUND

Use 'setoffset' to set the calibration offset of the I/O Module back to its default (0).

890406-02: **Incorrect calibration offset used for capture sync measurements.**

DESCRIPTION

When a measurement is made using the CAPTURE sync mode, the actual calibration offset value being used is the EXT sync mode calibration offset.

WORK AROUND

Set the EXT calibration offset to the desired CAPTURE calibration offset.

890406-03: **Setting capture offset value also sets EXT offset value.**

DESCRIPTION

When the CAPTURE calibration offset is set by 'setoffset', the EXT calibration offset is also set at the same time.

WORK AROUND

If both offsets need to be set, make sure that the CAPTURE calibration offset is set first, followed by setting the EXT calibration offset.

890413-01: **Startup program error on 9105.**

DESCRIPTION

"Disk corrupted. Object not a program." error message when the operator keypad RESET is pressed after first loading System Disk 1 and 2, System Disk 2 in /DR1 and User Disk in /DR2 (Start-up UUT and Pod database installed on User Disk). This same error message will occur when a new UUT with a different Start-up UUT

and Pod database is installed in /DR2 and RESET is Once Pod database is installed and program has been executed, you can press RESET and it will perform Start-up UUT correctly.

WORK AROUND

Load the UserDisk into and E-disk and RESET, or press execute keyto start the program.

890426-01: **ROM test displays ADDRESS STEP INVALID incorrectly.**

DESCRIPTION

ROM Test is done with an addr step of 5. When the test fails itdisplays the ADDRESS STEP INVALID message which is not correct.

WORK AROUND

None.

890504-02: **Log function causes read to choke.**

DESCRIPTION

The very simple program

```
program p
  log (10.0, 1.1)
  read addr 0
end program
```

causes an error in the read function "Illegal encoded space C0004" (or some closely related number). This was originally reported by a customer using the 68000 pod, but can be duplicated with an 80286.

WORK AROUND

1. Don't use floating point.

2. Try to put as much distance as possible between calls to floating point functions like log() and functions like read(). The problem is an internal variable whose value is not cleared. Putting some function calls (that happen to work) between the floating point function and the next pod function will tend to reset the internal variable.

890609-01: **TL/1 RESET function doesn't fully reset vector output I/O Module.**

DESCRIPTION

The TL/1 RESET function does not always fully reset a 9100-017 option. Depending upon the previous state, unpredictable measurements may result, or some of the output PINS may not be reset to TRISTATE. Also, relevant user programmable parameters are not set back to power-up values.

WORK AROUND

Press the RESET key on the Application Keyboard.

890615-01: **Print using doesn't handle null strings ("").**

DESCRIPTION

When the following TL/1 program is executed, various error messages unrelated to print are sometimes seen. The message "Badly formed format picture" should have been generated.

```
program tmp
    print using ""
end program
```

890626-02: **GOTO scope is incorrect.**

DESCRIPTION

If the same label name is used in multiple functions, sometimes TL/1 will jump to the wrong place when it sees a goto statement. It will jump out of the current function and goto that label in another function. The scope of the goto should be restricted to the function being executed.

To reproduce, execute the following program. It will generate the output:

```
Entering function F_1
Exiting function F_2
```

```
program gotobug

    function F_1
      print "Entering function F_1"
    L_6:
      if 10 > 5 then goto L_6
      print "Exiting function F_1"
    end function

    function F_2
      print "Entering function F_2"
    L_6:
      print "Exiting function F_2"
    end function

    F_10
end program
```

WORK AROUND

The bug is not exercised if you say 'end <name>' instead of 'end function'.

890711-06: **Copying programs to text files or ports uses up memory.**

DESCRIPTION

Certain editor operations, such as copying a UUT to a port resulted in an OUT OF MEMORY error message. It is possible the same error would result during a long editing session where large programs as well as other objects were edited, but this behavior was not observed.

WORK AROUND

There is no workaround for an operation that fails the first time. If the operation can be performed once, but not multiple times, memory can be reclaimed by exiting the editor.

6. Known Bugs In Version 4.2

870708-08: **TL/1 sometimes loads programs more than once.**

DESCRIPTION

Occasionally unusual disk activity was observed when a TL/1 program was executed in a loop. These programs were usually pod special programs.

The problem was that the program was named FOO but was invoked as foo. The operating system looks for the program on disk case insensitively, and when it finds the program, it executes it. However, it stored the program in a case sensitive symbol table, as required by the published semantics of TL/1. The next time the program was invoked, the symbol table was searched for foo. FOO was in the symbol table, but not foo, so the program was loaded again.

The bug is that the actual name of the program was not checked once the program was loaded.

WORK AROUND

The workaround for this bug is to correctly invoke the program FOO as FOO. That is, the name of a program must be spelled the same everywhere, and this spelling must match the name in the program statement.

No correct programs are broken.

This bug is left unfixed so that incorrect programs that 'seem' to work can continue to be used for awhile.

880122-08: **TL/1 Debugger can run out of stack space.**

DESCRIPTION

When repeatedly running a program from the debugger, the following error was issued: "Infinite Recursion or Stack Overflow"

The program being executed always generates a fault. The execution was being repeated with the following keys:

F10 (Fault)
F4 (Execute) <--- Started ANOTHER level of program.
Return

This behavior results because the original program is not complete. If F3 (CONT) was pressed instead of F4 (EXEC), the original program would continue. F4 (EXEC) causes a new copy of the program to start. Doing this enough times will fill the stack with multiple suspended copies of the program.

WORK AROUND

The INIT key discards all nested program invocations, clears all open file channels, reclaims RAM, and restarts TL/1's run time environment. Doing this will solve the problem for now, with the only side effect being that the breakpoints are erased.

880323-02: **Intermittent 9105 error when rebooting.**

DESCRIPTION

Intermittent 9105 error when rebooting

1. Reset 9105A without USERDISK
2. Install USERDISK in DR2
3. Hit Reset
4. Message - Podname does not match.....
5. Hit Reset
6. All OK

The correct Pod database is not picked up until the second reset.

Additional Info - if any disk is in drive 1 the problem goes away. Also DR1 light goes on if disk in drive.

WORK AROUND

Have a disk in Drive 1 when you reboot on 9105

880401-02: **Marking characters on long line can mess up attributes.**

DESCRIPTION

Bug involving MARKing characters on the bottom line of the screen so that the screen has to scroll. Line attributes (bolding) are wrong. To reproduce:

1) Edit a text file. Put a long line in it that takes up the rows on the CRT. Position it on the CRT so that the first row of the long line is on the bottom of the CRT. (The second row of the long line is off the screen.) Put the cursor on the first row of the long line.

2) Press MARK and -> until the screen scrolls and you've marked a few characters on the second row of the long line.

3) Press <- a few times. The bolding is not removed as the cursor backs up.

NOTE: The editor does YANK/CUT the correct # of characters, it's only the bolding that is messed up.

880425-02: **Editor positions cursor improperly on long lines.**

DESCRIPTION

Press <End Line> key on long line* or Press <Begin Line> and then <Return> - note changes example below

```
declare
   global string dvm = " 02"     !8840 on IEEE address 2
   global numeric ieee_out
   global numeric ieee_in
   global numeric disp
```

```
end declare

* ieee_out = open device "/port1", as "output", mode "unbuffered" !CR    print term
  ieee_in = open device "/port1", as "input"
  disp = open device "/term2", as "output"
```

880808-04: **Bogus error message when out of memory in TL/1.**

DESCRIPTION

I executed a large TL/1 program from the front panel (2 Mb hard model) and "repeat"ed it several times. At seemingly random times it would quit with a "NO DISK IN DRIVE" error message.

I just tried to repeat this without success. But this time the system behaved as though I had pressed "LOOP" because every time I exited the program, it would restart!

880815-01: **Deficiencies in DEFINE/TEST/SHOW RAM REF.**

DESCRIPTION

DEFINE/TEST/SHOW RAM REF

Has problem with ADDR option. SHOW does not show which option was used. TEST does not allow ADDR option to be shown or changed.

TEST sometimes comes up with illegal ADDR step message.

880909-04: **Application interface loses cursor.**

DESCRIPTION

1. Place the front panel cursor in rightmost field of the address option display (DWORD of 80386 MEMORY DWORD).

2. Execute a program that leaves the machine in a memory space with less fields in the address option. (I/O space of 80386)

3. Try to do a read or other function from the front panel. When

you go down to the address option field, the cursor disappears.

WORK AROUND

A left arrow key will make the cursor appear.

881005-04: **TL/1 has problems loading programs containing control codes.**

DESCRIPTION

A program which loaded into the editor without comment, passed the F10 "CHECK" function, and executed from the front panel caused the message,

Program '$debug$' could not be loaded properly.

when the F3 "DEBUG" key is pressed. Various versions of this program had run on the debugger earlier.

WORK AROUND

The program in question was written off-line and contained strings with control character codes that the 9100A editor does not insert. These 1 character codes were expanded into 3 character escape sequences which caused the end of the string to overwrite the next couple of characters. Thus although the program on disk had no syntax errors, the program saved in the debugger's temporary file did, and the debugger refused to load it.

Programs edited off-line must contain only printable characters.

881007-01: **Replacing single quote with double quote is not possible in programs.**

DESCRIPTION

On the following TL/1 line:

```
t = 't'
```

REPLACE ' with " and it worked for the first quote. Do a SHIFT REPLACE for the second quote and the error message:

String not closed by end of line. Missing double quote?

was displayed.

881019-04: **DEFINE/TEST RAM REF size problems.**

DESCRIPTION

Define multiple RAM references, some with DWORD size and addr increment of 4, some with WORD size and increment of 2, some with BYTE size and increment of 1. (can even be same space)

TEST RAM FAST ALL REF will return error of illegal step size.

881020-02: **Operator's i/f doesn't clear prompts.**

DESCRIPTION

Old prompts remain when new key is pressed.

Do a TEST BUS with 9132, (= PASSED appears) Press RAM key and select HYPER TEST (= PASSED may remain?) then ENTER The Hyper test prompts appear then press BUS key: The TEST BUS appears on the left with the old Hyper stuff on the right...

881028-03: **Bad display of READ DATA.**

DESCRIPTION

IF using the READ VIRTUAL function from the operator's interface, AND there are 12 significant digits in the EXTADDR and ADDR fields, AND there is 1 significant digit at that virtual address in the pod, THEN the data returned by the pod is not displayed on the 3-line display.

WORK AROUND

Write a TL/1 program to find the value. This case should NEVER happen. The VIRTUAL addresses have strong magic. Reading and writing from/to these address will most surely cause undue suffering.

881031-02: **Disk ID error in copy.**

DESCRIPTION

Using a 9100, format a disk from appl and then do a
COPY /DR1 to /DR1.
Then try to edit /DR1 type USERDISK immediately afterwards, and get the following message:
Disk ID error

WORK AROUND

One can eventually edit /DR1 by RESET-ing the system twice.

881102-01: **Restore allows incompatible Pods.**

DESCRIPTION

The 9100 allows RESTORE SYSTEM SETTINGS from settings which were stored using a different POD than the one curently in use.

Example: Power up with 8080 POD installed. Restores system settings from UUT or USERDISK where settings had been saved while using a 9132A-80386 pod. Message - Setup causes timeout. (Normal message when pod is in selftest). Now try setup pod. You can setup the number of ROM MODULES. (8080 POD does NOT have ROM MODULES, 9132 PODs do.)

NOTE: This action can cause strange system behavior. The only safe action after such an error is a full system REBOOT.

881104-01: **Overflow from addition not detected.**

DESCRIPTION

x = $FFFFFFFF + 1 overflows and should produce an error message. It doesn't. Overflow used to be detected.

WORK AROUND

This bug doesn't affect correct code.

881104-03: **Duplicate handler names not detected by TL/1.**

DESCRIPTION

There is no test in TL/1 to prevent the declaration of two handlers or two exercisers for the same fault in the same program.

```
program p
  handle h
  end handle
  handle h
  end handle
end program
```

This is obviously illegal and nonsensical but TL/1 doesn't test for it.

881104-04: **TEST BUS menu selection display disappears.**

DESCRIPTION

Perform TEST BUS from the front panel. Select another operation such as READ ADDR. Press the BUS TEST key. At this point the bus test softkey labels are missing. There should be two selections displayed.

WORK AROUND

Press BUS TEST key a second time, or press the left arrow key to get the selections to be displayed.

881104-06: **Editor COPY to port does not copy all of RESPONSE file data.**

DESCRIPTION

When using COPY in the editor to print RESPONSE files (copy the file to a port) the Priority Pin section of the file is not copied.

This means that there is no way for a user to get a printout of all of the information in a RESPONSE file.

Any other type of file, present or future, which uses multiple screens or more than 80 columns will probably demonstrate the same problem when printed.

881107-01: **V3.0 install disk programs do not run on V3.84.**

DESCRIPTION

With main software version 3.84 installed, the CONVERT and LIST utilities on the 3.0 Install disk do not work. This prevents use or even listing of TL/1 programs and data from old V2.x archive or backup disks that may not have been converted to V3.0 format because the disks were not currently in use.

If the CONVERT utility is run to convert a V2.x UUT to the new file format, all of the UUT files seem to be converted correctly except for PROGRAM files. The PROGRAM files are created in the new UUT, but attempting to edit these new files produces the error message "Disk corrupted! Not a program file."

If the LIST utility is run to list the contents of a V2.x userdisk, the utility halts after the Install disk is removed, with the error message
" List error code number =
49374".

WORK AROUND

Change the 9100A back to V3.0, and then run the CONVERT or LIST utilities. Finally, change back to the new version of software.

881109-04: **Editor display problems.**

DESCRIPTION

Using the up and down arrow keys in the editor, the screen gets hosed, with just a ">" shown in the left margin.

Sometimes when using the up and down arrow keys, lines are duplicated.

WORK AROUND

This appears to be related to the problem of displaying long lines (lines which wrap around). A work-around for both cases is to hit the INFO key, which forces the screen to redisplay properly.

881110-01: **Editor PASTE misbehaves if there are blanks at the end of the line.**

DESCRIPTION

It is not possible to PASTE at the end of a program line if the line ends in one or more blanks. Instead of the pasted text being put at the end of the line, it is inserted before the trailing blanks, and all of the blanks remain at the end of the line.

For example, on the following line, with the cursor at the end:

```
print "Test line"          <cursor is here>
```

PASTE text has this result:

```
print "Test line"!This is the pasted text          <End of line is here>
```

WORK AROUND

This problem is due to the editor's constant efforts to strip blanks off the ends of lines. Suggestion: paste in comments, then space/tab them out to the proper place.

881111-01: **Application display messed up after "infinite recursion" error.**

DESCRIPTION

TL/1 can be caught by surprise by a stack overflow which generally results from infinite recursion in a function or a fault handler. The TL/1 process dies very abruptly and returns control to the operator's interface.

The operator's interface displays the message "Infinite Recursion or Stack overflow". If you press ENTER to restart the program, the pod interface may be in an inconsistent state. Among the error messages seen is "Internal error 8046: Timeout mode set". LOOP and REPT are also not usable.

WORK AROUND

Restarting the operator's interface gets things back to normal. It is often possible to re-enter the command (all keys, not just ENTER) and have it execute normally.

881111-02: **Noise on RS232C lines causes program error.**

DESCRIPTION

input statement using buffered channel opened to /port1 causes error message "Read Error". The probable cause of the error is noise on /port1 caused by a 5-to-1 terminal switch, and also by powering up the 9100, since the problem occurs the first time the program is run after powering up the 9100 or changing the switch, but not subsequently.

WORK AROUND

executing a poll(channel, "error") clears the error status and avoids this problem. This may be the fix.

881114-01: **9100 cannot always tell if pod database name does not match pod type.**

DESCRIPTION

Plug in a Z80 pod (with the UUT cable in the selftest socket), and enter a POD NAME of 6802. The 9100 will load the pod database, and return the error message "enabled line causes timeout" (which is normal when the pod is in selftest). The pod database loaded now is for the 6802 pod (as evidenced by the ADDR OPTION, ENABLE line names, and POD NAME), however the pod will pass selftest and seems to work normally for other functions as well.

Selftest, at least, should be able to tell that the pod type does not match the database that is loaded, since the pod reset string includes its name. The POD NAME entered should also be checked against the actual pod installed.

This seems to happen specifically with the Z80 pod and the 6802 database. Other combinations of pod and database mis-matches do give an error message if the wrong name is entered, but the mis-matched database is still loaded.

881212-03: **Debugger SHOW and SET do not work for array references.**

DESCRIPTION

SHOW and SET in the debugger do not work for array references:

- o Array indices cannot be specified in SHOW or SET.
- o SHOW and SET can be used with just the array name which appears to show the lowest subscripted element.

WORK AROUND

1. Don't attempt to SHOW or SET the zero'th element of the array. It may crash the debugger.

2. Array elements can be assigned to a variable (in the program) and that variable examined with SHOW. It may also be convenient to write a small function to print the entire array contents (if the array is declared global).

890111-01: **Errors in 68000 part description.**

DESCRIPTION

The list of related input pins (beside the IC picture) for the 68000 were typed in the "Pin Name" field. They should be in the "Related Input Pins" field.

Below the IC there are more lines of "Related Input Pins" which are incomplete. They should be deleted (the data is specified beside the IC picture.

WORK AROUND

Edit the 68000 part description. Beside the IC picture, move all occurrences of "0" from the "Pin Name" field to the "Related Input Pins" field. Delete lines 77 - 100.

890124-02: **Pod program QWK_RAM has syntax error.**

DESCRIPTION

8088MX pod, 9100A under 3.0 software.

When using the 8088 pod in max mode, the program QWK_RAM did not work due to a syntax error. The QWK_RAM in the 8088 pod file is OK.

WORK AROUND

An "end if" statement has to be taken out. Remove the "endif" line 4 lines above the "end QWK_RAM" statement at the end of the program.

890202-03: **Pasting lines from text file into a program makes screen incorrect.**

DESCRIPTION

Pasting lines of text (non-TL/1) into a TL/1 program screws up the editor screen. What you see is not what is there.

WORK AROUND

Pressing INFO repaints the correct stuff.

890227-01: **Using HELP in the E-DISK LOAD command can be non-intuitive.**

DESCRIPTION

When choosing a UUT to load into the E-DISK, the user (as per normal) can push the HELP button when in the UUT NAME field of the E-DISK load command on the application interface, and get a list of the UUTs on the "current disk". However, if the user sets the DRIVE field in the E-DISK LOAD command to a disk different than the "current disk" (as set in the SETUP command), the command still lists out the contents of the "current disk". It would be intuitive that it should list out the contents of the disk that the user chose in the E-DISK LOAD command. This can be confusing.

WORK AROUND

Go to the SETUP area and SETUP USERDISK to the drive that is being loaded into the E-disk.

890308-02: **Debugger chokes on stop signal and dies.**

DESCRIPTION

Consider the program fragment

```
program a
   execute large_program()
   ...
end a
```

In the debugger, if you execute program a, then press the QUIT key during the time large_program is being loaded from the disk, then ask to SHOW an undeclared name, there is a brief flurry of disk activity, then the debugger exits suddenly and the application display shows the message

OBJECT DOES NOT EXIST

WORK AROUND

Don't press QUIT while the disk light is lit.

890309-03: **Errors in help files.**

DESCRIPTION

The TL/1 help files (reached by pressing the HELP key while in the editor) have the following errors:

UUT help

The SEARCH softkey used in the help file is F10; it should be F9

The F3 COMPILE softkey description does not mention the "TROUBLESHOOT UUT" or "LEARN RESPONSES" field selection, only the "GFI" or "UFI" field

PROGRAM help

No description of info window, editing, softkeys, etc., like are found in the help windows for all other file types

TEXT, NODE help

The DESCRIPTION field in the info window is described but does not exist (this may be a bug in the editor and not a help window error)

TEXT, RESPONSE, NODE, REF, PART help

The WRITE PROTECT field in the info window is not described

RESPONSE help

The F8 MORE softkey is not described

The field in the info window that is called REPS in the help description is actually named REPEAT

REF, PART help

The F6 softkey is not DELETE; the F7 softkey is not INSERT

The F4 MARK, F5 CUT, F6 YANK, and F7 PASTE softkeys are not described

PARTLIB, PODLIB, PROGLIB, HELPLIB help

The F6 softkey is not EDIT; the F7 softkey is not QUIT

PROGLIB, HELPLIB help

The F6 STYLE softkey is not described

POD help

The PODDB file type is not mentioned; it should say that they can't be edited

DEBUG help

The description of the F7 SHOW softkey says that number values are displayed in hexadecimal (decimal for shift-F7); actually, number values are displayed in decimal (hexadecimal for shift-F7)

The TL/1 help files also have the following minor problems:

USERDISK help

The description of the NAME field in the info window says the NAME is "/HDR" or "/DR1"; actually, the NAME is displayed without the initial "/"

The DISK PROTECTED field in the info window is not mentioned

RESPONSE help

In the help text, it says "Changes in the info window will require a YES..."; this should say "Changes in the info window or text will require a YES..."

In the Priority Pin description, the word "override" is misspelled

PORT (TERM) help

In the help text, it says "Press QUIT to return to the UUT directory."; but, this is incorrect when TERM is started from a USERDISK directory

WORK AROUND

Take the help window information with a grain of salt.

890313-02: **Error message problems when booting from non-validated hard disk.**

DESCRIPTION

When attempting to boot up from the hard disk when the serial number of the 9100A does not match the serial number saved on the disk, the error message

This machine is not validated to run system software on the hard disk.

is displayed for about a second, and then is overwritten with the

message

root: Ramdisk initialization failed Can't find the error table!
-- PRESS ANY KEY --

When a key is pressed, the display message changes to "root: Starting " and the 9100A seems to hang with no response.

This could be a serious problem for field service, since the real failure information is overwritten by an unnecessary error message. The only time this bug is likely to be seen would be as a result of a hardware failure.

890314-01: **Readblock, writeblock, and loadblock defaults are not the same from the front panel as from TL/1.**

DESCRIPTION

The default file format for readblock, writeblock, and loadblock is "motorola". Unfortunately, when started from the front panel, the default file format is "intel". For compatibility, this must be changed to "motorola".

890316-01: **Testromfull using mask incorrectly.**

DESCRIPTION

When I did a testromfull from TL/1 with a mask of $2000, it should have only reported faults on bit 13. However, it was reporting faults on bits 0 - 13.

TL/1: testromfull addr $F0000, upto $FFFFE, mask $2000, step 2

890321-01: **Memory disappears in Bus Test.**

DESCRIPTION

Select bus test on the front panel. If the address line stuck fault appears and you loop on the fault, memory leaks away. It takes about 45 minutes to exhaust 4Mbytes of RAM. Other faults reported by bus test may have the same problem but other tests seem unaffected.

WORK AROUND

Select BUS_TEST ENTER LOOP to restart loop.

890331-01: **Hyper-Help bug with greater than 105 files.**

DESCRIPTION

If there are more than 105 of a given object when hyper-help is invoked to select a file from that object type, Hyper-Help beeps and displays only 105 of the files. You may then scroll past the end of the files, lose the cursor, and never get it back. The No/Clear button exits Hyper-Help correctly.

WORK AROUND

Press the No/Clear button and type any programs that are not displayed by hand.

890331-02: **Stop key won't stop TL/1 program.**

DESCRIPTION

The STOP and reset keys will not stop a TL/1 program during the time it is executing a loop consisting of only the LOOP and END LOOP statements. For instance, to hang the 9100 beyond recovery execute the four line TL/1 program

```
program hang
  loop
```

```
end loop
end program
```

WORK AROUND

Adding any statement (except another loop) inside the loop permits the keys to be recognized.

890403-01: **Variable-width formated input does not detect invalid input.**

DESCRIPTION

Variable-width formatted input (input using "?%") should accept *1* or more hex characters to be input. It allows 0 hex characters (an invalid hex number) to be entered and does not complain.

890405-01: **CADNETIX example in Programmer's Manual bad.**

DESCRIPTION

The CADNETIX example in Chapter 7 (CAD Translator) of the programmer's manual is wrong. Line 5 reads:

```
R1206V,200K          X   R20 R21 R22 R23 R24
```

It should read:

```
R1206V,200K          X   R20 R21 R22 R23 R24 R10
                                 ^^^^
```

Line 13 reads:

```
C10 1 C13 1 C15 1 R20 1 R23 1 U1 $
```

It should read:

```
C10 1 C13 1 C15 1 R20 1 R23 1 U1 8 $
                        ^^
```

These cause the example to not be able to run through CADTrans.

I have given markups to Pat East.

890412-01: **Pod configuration file not downloaded upon pod reset.**

DESCRIPTION

Found while working with a Jane pod. To duplicate, unplug the pod from the mainframe (also turn off the the uut to protect the pod.) Then plug it back in. This will cause the pod to timeout and the mainframe will reset the pod when the mainframe tries to command the next operation. When the mainframe resets the pod, it should restore the state of the pod, including the config file. But the config file is not restored! Run the pod-specific program "VERSION" to show that the pod has not been given a config file. Perform the workaround given below to see what the "VERSION" program should display when the pod truly has received a config file.

WORK AROUND

Use the SETUP key on the operator's keyboard to set the POD NAME. This will force the mainframe to download the config file. Usually one must RESTORE the system settings for the current UUT to counter some side effects of SETUP POD NAME. Confirm that the pod has received a config file by running the "VERSION" program. The program will display a version number for the config file which was downloaded.

890418-01: **TL/1 mid function dies with length argument of zero.**

DESCRIPTION

More than once now I've had string processing algorithms broken because "mid" pukes if you give it a length argument of zero. I think it is perfectly reasonable to ask for a zero length string and have the function return just that. Here is an example of the kind of thing which fails,

```
                                ! Change all z's to Z's
loop for i = 1 to len(x)
   if mid(x,i,1) = "z" then
      x = mid(x,1,i-1) + "Z" + mid(x,i+1,len(x)-i)
   endif
end loop
```

Notice that this will fail if a "z" appears as the first or last character in "x". It may also fail because the start position is too high, and that, I concede, makes this a poor example. But you should get the idea.

WORK AROUND

Use a different algorithm to avoid giving mid zero as a length argument.

890419-01: **Setting time on 9105's with no real time clock.**

DESCRIPTION

If a 9105 does not have a real time clock, and the time is set to a valid value via the front panel, the 9105 will hang, and require rebooting. If the 9105 has a real time clock, everything works just fine.

890501-01: **FutureNET translate bug in CADTrans.**

DESCRIPTION

AT&T uses a (DATA statement after the (SIG statement and before the (NAME statement. FutureNET incorrectly throws away these net names.

This is the description that he gave to me. The investigator will have to figure out what this means.

WORK AROUND

Remove the (DATA statement from the FutureNET netlist.

890504-01: **Error in 80186 quick ROM test.**

DESCRIPTION

Line 107 of 80186/QWK_ROM contains a readspecial command, only it is misspelled readspecail.

WORK AROUND

Change readspecail to readspecial.

890515-01: **Problems with debugger.**

DESCRIPTION

1. If you are at the last statement of a function and press STEP (F1), the debugger won't necessarily single step. It may continue.

2. If you are at the line before a function that will raise a fault, you might think STEP (F1) would stop you on the first line of the fault handler. It doesn't.

WORK AROUND

1. What happens is that if you did a STEP (F1) to get into the function, when the function returns you'll be in STEP mode, but if you got into the function using a breakpoint, the caller will be in CONT mode.

2. The way to stop in a handler is to set a breakpoint at the first line of the handler.

890518-01: **It is illegal to initialize global variables.**

DESCRIPTION

It is illegal to initialize global variables in the declaration statement, but TL/1 does not detect this. The following problems can occur if you initialize global variables.

1. If global variables are initialized in multiple places, only the first such declaration has effect. Thus if you intend an initialization to clear the value of a global variable, this will not happen.

2. If you loop or repeat a program, or execute it multiple times from the debugger without using the INIT key between executions, the initialization will not take effect on the second and subsequent invocations. Thus the program

```
program p
   declare global x = 0
   x = x + 1
   print x
end p
```

will print 1, 2, 3, etc. on succesive loop iterations or F4 (EXEC)'s.

WORK AROUND

Do not attempt to initialize global variables. use an assignment statement to initialize them in the places where initialization is desired.

The manual states that it is illegal to initialize global variables. The manual is correct.

890609-02: **RAMTEST doesn't detect address fault with address Step of 6.**

DESCRIPTION

With the 82086 and Demo Trainer, tie A1 or A2 to ground. RAMTEST FULL ram with an address step of 6. Observe that no fault is reported.

WORK AROUND

Use an address step that is a power of 2.

890614-01: **Sometimes 'gfi status' incorrectly returns a status of "untested".**

DESCRIPTION

If you execute a TL/1 program that calls 'gfi status' after running GFI from the front panel (via GFI-RUN menu), then 'gfi status' always returns a status of "untested". It doesn't seem to know about the pins which were tested from the front panel.

890620-01: **ROM TEST address step is in HEX on front panel.**

DESCRIPTION

Is it a bug that the address step for a front pannel ROM TEST is in hex? I thought all default valuse changed to decimal at 3.0.

I think this is true for RAM TEST FAST also.

890620-02: **4.0 QW_RAM program for 80186 pod is incorrect.**

DESCRIPTION

The QWK_RAM program of pod 80188 is missing the call to pretestram

890626-01: **TL/1 'testbus' and front panel TEST BUS raise different faults.**

DESCRIPTION

Using a Z80 pod, when an address line is tied low, the front panel TEST BUS raises the 'bus_addr_low_tied' fault ("addr line A0 pin 30 stuck low" message on the display), but the TL/1 'testbus' command raises the 'pod_addr_tied' fault instead ("addr line A0 pin 30 not drivable" message). When an address line is tied high, both TEST BUS and 'testbus' seem to raise the same fault, 'bus_addr_high_tied'.

One effect of this is that a fault handler for 'bus_addr_high_tied' will handle address lines tied high during 'testbus', but a fault handler for 'pod_addr_tied', rather than 'bus_addr_low_tied', is required to handle address lines tied low.

WORK AROUND

Add a fault handler for 'pod_addr_tied' that either raises the correct fault or duplicates the 'bus_addr_low_tied' fault handler.

890627-01: **68000L pod database bug.**

DESCRIPTION

68000L pod database does not parse correctly.

890711-02: **MORE INFORMATION LED left on incorrectly.**

DESCRIPTION

When doing a TEST_BUS with the 9132A POD, and a fault message causes the MORE INFORMATION LED to come on, the LED does NOT go off when the CONT key is pressed, even though the new message fits in the window.

The actual fault was BAD CHIP SELECT on the ROM MODULE, cause by tied high W/R line.

890711-03: **PASSED message is not erased corectly.**

DESCRIPTION

After doing a bus test on a 9132A POD, you have to press the BUS TEST key twice to erase the PASSED message. A single press of another key such as the down arrow will erase the message. I believe a single press of BUS TEST used to erase the PASSED. (Repeat never did).

890711-04: **Displayed data truncated in READ VIRTUAL.**

DESCRIPTION

The displayed data is truncated when READ VIRTUAL is performed from the operator's interface, if the EXTADDR and ADDR values have more than 8 digits total. For example, if the data value read is $FFFF,

READ VIRTUAL EXTADDR 2000000 ADDR 0 = FFFF

will be displayed correctly, but

READ VIRTUAL EXTADDR 2000000 ADDR 64 = FFF

is truncated at the right hand edge of the display. This means that the whole data value (of 16 bits) cannot be read by the operator.

890717-01: **I/O should be flushed on program exit.**

DESCRIPTION

The following program displays no output on termination.

```
program p
  print using %", 1
end p
```

Should TL/1 flush output when the program ends?

890718-01: **Error message doesn't display argument name.**

DESCRIPTION

```
program foo
    function bar
      return
    end function

    bar blah 1
end program
```

In this program, function bar is called with an illegal argument 'blah'. The error message that results is "'' is not a legal argument." The name 'blah' should appear between quotes.

890718-02: **Sign of negative floating point numbers is printed in wrongcharacter position.**

DESCRIPTION

The TL/1 statement
  print using #^^^^^.^^^", -1.19
should print "-00001.190". Instead, it prints "000-1.190".

890727-01: **Infinite recursion or stack overflow error in UUT compiler.**

DESCRIPTION

When compiling a UUT, an "Infinite Recursion or Stack Overflow" error occurred.

WORK AROUND

This can occur if the reflist contains several hundred entries and they are in alphabetical order. To workaround this, move the middle entry to the beginning of the reflist and recompile.

For example, if the reflist is 600 lines long, move the entry at line 300 to line 5 (lines 1-4 of the reflist are title lines).

7. Manual Update

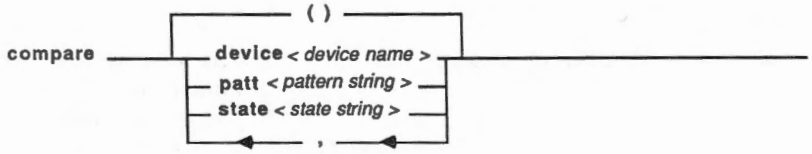
Several new functions were added to TL/1 for release 4.1. The following pages are manual updates and corrections for the *TL/1 Reference Manual.* These manual pages are numbered in a new way. "Page numbers" now have the form "name-number", as in "read-1" or "TestRAMfast-3". Insert these pages into section 3 (the *TL/1 Alphabetical Reference* section) of your manual, in their alphabetical order. A new table of contents and index will be provided as part of software release 5.0.

No Text on This Page

Syntax:

```
compare [device <device name>] [, patt <pattern
     string>] [, state <state string>]

compare (<device name>, <pattern string>, <state
     string>)

compare ()
```

Syntax Diagram:



Description:

Monitors pins on an I/O module for the occurrence of the specified pattern and generates a "iomod_dce" fault condition when a match occurs.

Arguments:

device name	I/O module name, or clip module name. (Default = "/mod1")
pattern string	String expression for the comparison pattern. The left-most character in the pattern string corresponds to pin 1. (Default = "1")
state string	"enable" or "disable" comparison. (Default = "enable")

Example 1:

```
compare device "/mod2", patt "1011100XXXX11X",
 state "enable"
```

Example 2:

```
compare ("/mod1A", "1111111111111111", "disable")
```

Example 3:

```
program test9
     .
     .
     handle iomod_dce        ! This handler is called
                             ! whenever the bit
                             ! pattern specified in a
         print "successful DCE"! compare command
                             ! matches the pattern on
                             ! pins being monitored
                             ! by an I/O module
     end handle
     .
     .
     compare device "/mod1", patt "10111XXX001",
     .
     .
end program
```

Remarks:

A fault condition is generated when the specified bit pattern is detected. The *compare* command might be used to generate a fault condition when a particular UUT address is accessed or when particular data is on the data bus.

The pattern is specified as a single string that may contain the following characters:

1: high
0: low (or invalid levels)
x or X: don't care

Note that both low levels and invalid levels are considered to be low when making comparisons.

The characters in the string are mapped onto pins with the left-most character corresponding to pin 1. For example, if you want to detect when pins 1, 2, and 3 are high and when pins 12, 13, and 14 are low (and you do not care about the state of pins 4-11), you specify the pattern "111XXXXXXXX000".

You can specify the pattern in terms of clip pins or I/O module pins. If the device has more pins than the pattern, the excess pins are ignored.

The *compare* command may be used to compare a pattern string with up to 40 pins if they are all on the same I/O module and depending on what clip module is used.

The data compare equal (DCE) condition is detected as soon as it occurs, but the resulting DCE fault condition is raised only when an I/O module or probe function is executed or when a pod-access function is executed.

For More Information:

- The "Overview of TL/1" section of the *Programmer's Manual.*

Syntax:

```
cwd ()
```

Syntax Diagram:

Description:

Returns the current working directory as a string.

Returns:

The current working directory is returned as a string. If program execution began in a UUT directory, *cwd* returns a string of the form "/userdiskname/uutname" as in "/HDR/ABC". If execution began in the podlib, *cwd* returns a string of the form "/userdiskname/PODLIB/podname" as in "/HDR/PODLIB/80286". If execution began in the proglib, *cwd* returns a string in the form "/userdiskname/PROGLIB", as in "/DR1/PROGLIB".

Example:

```
d = cwd()
if instr (d,"PODLIB") or instr (d,"PROGLIB") then
  print "Current directory is not a UUT".
end if
```

Remarks:

The current working directory is the directory from which program execution began, not the directory of the program currently being executed. This distinction is important for programs in the proglib and podlib.

The string returned by *cwd* can be useful when constructing absolute file names for text files.

Related Commands:

filestat, open, close

For More Information:

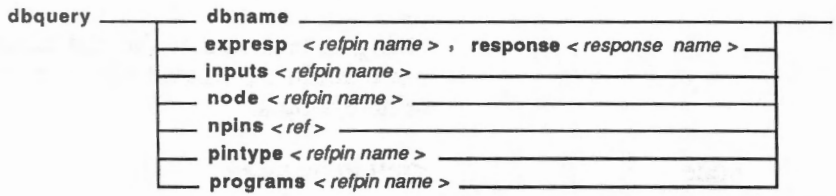
- The "Overview of TL/1" section of the *Programmer's Manual.*

dbquery

Syntax:

```
dbquery dbname

dbquery expresp <refpin name>, response <response
     file name>

dbquery inputs <refpin name>

dbquery node <refpin name>

dbquery npins <ref>

dbquery pintype <refpin name>

dbquery programs <refpin name>
```

Syntax Diagram:



Description:

The *dbquery* commands allow a TL/1 program to retrieve information from the Compiled UUT Database.

Options:

dbname — (Has no argument value)

This option returns a string containing the name of the UUT compiled database ("GFIDATA" or "UFIDATA"). If the UUT does not contain a compiled database, an empty string is returned.

expresp <refpin name>, response <response file name>

This option returns the expected response data for the named pin and stimulus program. The data is returned as a comma-separated list of "type=value" pairs, where type is "sig", "alvl", "clvl", or "count", and value is the data that appears in the response file. The list will only contain the types of response data that will be used to compare.

inputs <refpin name>

This option returns a string containing a comma-separated list of related input pins for the named pins. An empty string is returned if the database does not contain the named pin or it has zero related inputs.

node <refpin name>

This option returns a string containing a comma-separated list of pins that are members of the same node as the named pin. If the database is for UFI (which does not use a nodelist), an empty string is returned. If the database is for GFI and the named pin did not appear in the nodelist, an empty string is returned.

npins <ref>

This option returns a string containing the number of pins on the named reference designator. If the database does not contain the named reference designator, an empty string is returned.

pintype	<refpin name> This option returns a string identifying the pin type of the named pin ("INP", "OUT", "BID", "PWR", "GND", or "UNU"). An empty string is returned if the database does not contain the named pin.
programs	<refpin name> This option returns a comma-separated list of TL/1 stimulus programs that will be used to test the named pin. This includes programs that test the pin as an input and programs that test it as an output. If the database does not contain the named pin, or if the database does not describe how to test the named pin, an empty string is returned.

Example 1:

```
        ! print the list of pins that are on the same
        ! node as U1-25
print "node = ", (dbquery node "U1-25")
```

Example 2:

```
        ! print the number of pins on R33
n = dbquery npins "R33"
print "R33 has ", n, " pins"
```

Example 3:

```
        ! This example prints the list of programs
        ! that are used to test U1-b4
list = dbquery programs "U1-b4"
print "U1-b4 is tested by the following programs: "
  , list
```

Remarks:

The compiled UUT database is used by the resident GFI software. The 9100A/9105A will automatically load the database off disk and into memory the first time a *dbquery* command is executed. However, you must use the *gfi clear* command to unload the database when you are finished accessing it. Failure to do so will decrease the amount of memory available to the 9100A/9105A.

Refer to the *Programmer's Manual* for a description of GFI and for information on how to create a UUT database for GFI.

Related Commands:

gfi clear

For More Information:

- The "Guided Fault Isolation (GFI)" section of the *Programmer's Manual*.

Syntax:

```
filestat file <file name string>

filestat (<file name string>)
```

Syntax Diagram:

Description:

Returns information about the existence, readability, and writeability of a text file.

Arguments:

file	A string containing a relative or absolute file path name.

Returns:

A three character string if the file exists, or the empty string if the file does not exist.

The first character of the string is "r" if the file is readable and "-" otherwise.

The second character of the string is "w" if the file is writable and is "-" otherwise.

The third character of the string is "-".

Example:

```
print DEMO, print "DEMO--", filestat ("DEMO")
```

The example prints "**DEMO--rw-**" if the text file DEMO in the current UUT exists and is not write protected.

Remarks:

Filestat returns information about the status of the file. It cannot tell if a floppy disk write-protect tab is in the protected position.

Related Commands:

cwd, open, close

For More Information:

- The "Overview of TL/1" section of the *Programmer's Manual.*

getpod

Syntax:

```
getpod podname
```

Syntax Diagram:

Description:

Returns information about the current pod.

Arguments:

podname	Return the name of the currently connected pod.

Returns:

The name of the current pod.

Example:

```
if instr ((getpod podname), "M") = 1 then
    print "9132A pod in use"
end if
```

Remarks:

There is a current pod name (called "32BIT") when no pod is plugged in. 9132A pod names all begin with "M", so getpod is useful to determine the type of pod.

Related Commands:

podinfo, podsetup

For More Information:

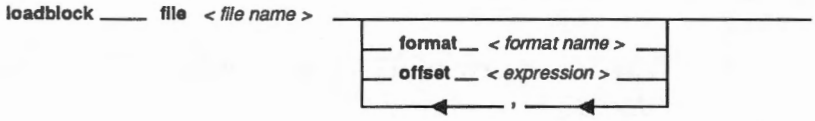
- The "Overview of TL/1" section of the *Programmer's Manual*.

loadblock

Syntax:

```
loadblock file <file name> [, format <format name>,
     offset <exp>]

loadblock (<file name>, <format name> , <offset
     expression>)
```

Syntax Diagram:



Description:

Loads the contents of a file in a standard ASCII form (Motorola S-Record format or Intel Hex format) into UUT or pod overlay RAM. The file contains information about the starting address and number of data bytes.

The value of the offset expression is interpreted as a 32-bit 2's complement value and added to the address in each record of the data file, to obtain the load address.

Arguments:

file name	The name of the file containing the required data.
format name	The ASCII format in which the data was previously stored. Either "intel" or "motorola". (Default = "motorola".)
offset	This value is added to each address in the data file to obtain the load address. (Default = 0.)

Examples:

```
loadblock file "test_lcd", format "motorola"

loadblock ("pgm1", "motorola", $10000)
```

Related Commands:

readblock, writeblock

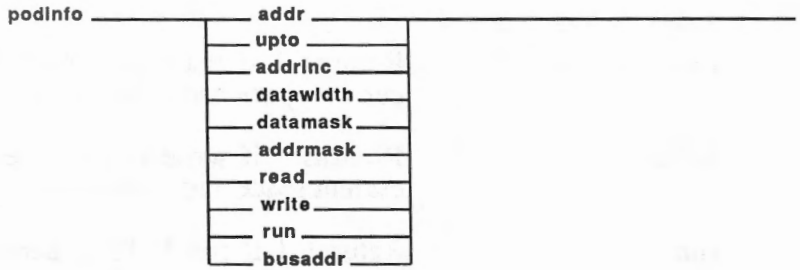
For More Information:

- The "Overview of TL/1" section of the *Programmer's Manual.*

Syntax:

```
podinfo addr
podinfo upto
podinfo addrinc
podinfo datawidth
podinfo datamask
podinfo addrmask
podinfo read
podinfo write
podinfo run
podinfo busaddr
```

Syntax Diagram:



Description:

Returns the requested information about the current space.

Options:

addr	Returns the lowest valid address in the current space. All valid addresses in the current space are greater than or equal to this number.
upto	Returns the highest valid address in the current space. All valid addresses in the current space are less than or equal to this number.

addrinc	Returns the minimum valid address increment in the current space. All valid address increments are multiples of this number.
datawidth	Returns the width, in bits, of the data words in the current space.
datamask	Returns the bitmask with bits set for the valid data bits in the current space.
addrmask	Returns a bitmask of valid address bits. Note the least significant couple of bits may not be set if addrinc is greater than one.
read	Returns 1 if read is permitted in the current space and 0 otherwise.
write	Returns 1 if write is permitted in the current space and 0 otherwise.
run	Returns 1 if run UUT is permitted in the current space and 0 otherwise.
busaddr	Returns the default bus test address.

Returns:

The requested numeric value is returned.

Examples:

```
function testme (addr, upto)

     if addr < podinfo addr then fault\return
     if upto > podinfo upto then fault\return

     testramfast addr addr, upto upto,
      addrstep 2

end function
```

Related Commands:

setspace, getspace, sysspace, podsetup, sysaddr, sysdata

For More Information:

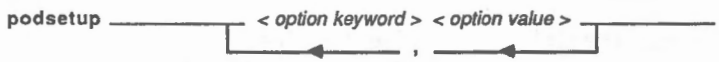
- The "Overview of TL/1" section of the *Programmer's Manual.*
- *Supplemental Pod Information for 9100/9105A User's Manual.*

podsetup

Syntax:

```
podsetup { <option keyword> <option value> }
```

Syntax Diagram:



Description:

Accesses the pod to enable or disable reporting of faults directly sensed by the pod hardware.

Arguments:

Option Keyword	*Option Value(s)*
'report power'	"on" or "off". Enables or disables reporting of bad power supply level.
'report forcing'	"on" or "off". Enables or disables reporting of active forcing-input lines.
'report intr'	"on" or "off". Enables or disables reporting of active interrupt lines.
'report address'	"on" or "off". Enables or disables reporting of undrivable address-output lines.

'report data'	"on" or "off". Enables or disables reporting of undrivable data bus lines.
'report control'	"on" or "off". Enables or disables reporting of undrivable control-output lines.
'report special'	"on" or "off". Enables or disables reporting of special pod errors.
'enable string'	"on" or "off". Enables or disables a pod-dependent forcing line. The enable phrases all begin: 'enable ' and end with a pod-dependent string.
timeout	<expression> Changes the timeout time. The expression must be numeric.
option	<value> A pod-dependent setup option. The option and value are defined by the pod data file.

Example 1:

```
podsetup 'report power' "on"
```

Example 2:

```
podsetup 'enable ready' "off"
```

Example 3:

```
podsetup timeout 1000
```

Example 4:

```
podsetup 'report intr' "off", 'report power' "off"
```

Remarks:

The *podsetup* function may be written in keyword notation only.

For More Information:

- The "Overview of TL/1" section of the *Programmer's Manual.*
- *Supplemental Pod Information for 9100A/9105A Users Manual.*
- The Fluke pod manual for the microprocessor you are using.

random

Syntax:

```
random [seed <expression>]

random ([<expression>])
```

Syntax Diagram:

Description:

Produces pseudorandom sequences of numbers.

Arguments:

seed	Providing this optional value changes the sequence of numbers.

Returns:

Returns a pseudorandom 32-bit numeric value.

Example:

```
loop for i = 1 to 100 ! print 100 random numbers
    print random()
end loop
```

Remarks:

The sequence of numbers returned by random has the appearance of randomness, but no specific property of the sequence is guaranteed. The sequences of numbers generated by *random* may change in subsequent software revisions to provide more nearly random sequences.

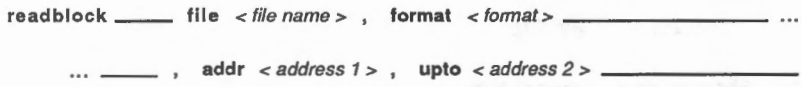
For More Information:

- The "Overview of TL/1" section of the *Programmer's Manual.*

Syntax:

```
readblock file <file name>, format <format>,
     addr <address 1>, upto <address 2>

readblock (<file name>, <format>, <address 1>,
     <address 2>)
```

Syntax Diagram:



Description:

Reads the data from the specified address range and stores this data in the specified text file.

Arguments:

file name	The name of the file in which to store the data. If a full path name is not specified, the data will be stored in the specified file in the current UUT directory.
format	The ASCII format in which the data was previously stored. Either "motorola" or "intel". (Default = "motorola".)
address 1	Start address.
address 2	End address.

Example 1:

```
readblock file "testlcd", format "motorola",
 addr $7000, upto $7FD4
```

Example 2:

```
readblock ("pgm1", "motorola", $1000, $1FFF)
```

Example 3:

```
readblock file "/HDR/testdata", format "motorola",
     addr $7000, upto $7FD4
```

Related Commands:

loadblock, writeblock

For More Information:

- The "Overview of TL/1" section of the *Programmer's Manual.*

readvirtual

Syntax:

```
readvirtual extaddr <extended address>, addr
    <address>

readvirtual (<extended address>, <address>)
```

Syntax Diagram:

Description:

Readvirtual is a complete replacement for the obsoleted *readspecial* command. Returns the data located at the specified virtual address. This allows access to the virtual addresses that, in some pods, are used for special operations. This command should only be used when you know that the normal *read* command does not provide the required special operation.

Arguments:

extended address	Upper 32-bits of virtual address.
address	Address from which to read data.

Returns:

The data at the specified address.

Example 1:

```
value = readvirtual extaddr 0, addr $F0000018
```

Example 2:

```
value = readvirtual (0, $F0000018)
```

Remarks:

Incorrect use of this special-purpose command can place the pod and the 9100A/9105A mainframe in inconsistent states.

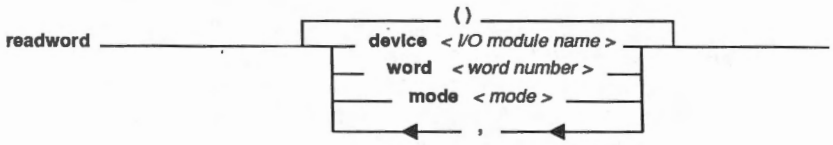
For More Information:

- The "Overview of TL/1" section of the *Programmer's Manual.*
- The Fluke pod manual for the microprocessor you are using.

Syntax:

```
readword [device <I/O module name>] [, word <word
    number>][, mode  <mode>]

readword(<I/O module name> , <word number> , <mode>)
```

Syntax Diagram:



Description:

Allows a group of I/O module pins to be read as a single group of values.

Arguments:

I/O Module name	I/O module name ("/mod1", "/mod2", "/mod3", or "/mod4"). (Default = "/mod1")
word number	The number of the word group. Valid values are from 1 through 5. (Default = 1)
mode	The mode in which to read the word. Valid values are "now" or "stored". (Default = "now")

Returns:

A string representing the state of the pins specified by the *setword* function.

Example 1:

```
! Read the current level of pins 1-4 of I/O Module #1
setword device "/mod1", word 1, as_pins "1 2 3 4"
binstr = readword device "/mod1", word 1, mode "now"
```

Example 2:

```
! Read the stored level of pins 1-4 of I/O Module #1
clockfreq device "/mod1", freq "1MHZ"
edgeoutput device "/mod1", start "at_vectordrive"
sync device "/mod1", mode "capture"
syncoutput device "mod/1", mode "intfreq"
setword device "/mod1", word 1, as_pins " 1 2 3 4"
vectorload device "/mod1", file "demo"
arm device "/mod1"
  vectordrive device "/mod1", startmode "now"
readout device "/mod1"
binstr = readword device "/mod1", word 1, mode "stored"
if instr(binstr,"*") = 0 then
  wordone = val(binstr,2)
else
  wordone = 0
end if
```

Remarks:

In the "now" mode, the *readword* function mimics the operation of the INPUT WORD operation from the front panel. The current logic level of the pins in the word are grouped into a string. In the "stored" mode, the *readword* function takes the clocked level history information from the values that are stored in memory from the latest readout command.

A pin level of HIGH is represented as a logic "1". A pin level of LOW is represented as a logic "0". Any other pin level is represented as an "*", so use the *level* function to determine the actual level.

Setword can make pin grouping assignments. Groupings may be made from 1 to 40 pins.

Related Commands:

setword, writeword

For More Information:

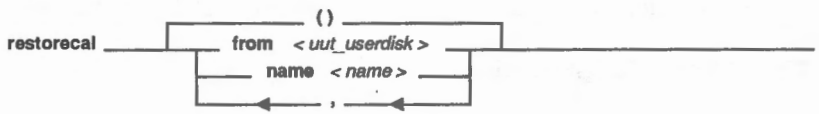
- The "Overview of TL/1" section of the *Programmer's Manual.*

restorecal

Syntax:

```
restorecal [from <uut_userdisk>] [, name <name>]

restorecal (<uut_userdisk>, <name>)

restorecal ()
```

Syntax Diagram:



Description:

Restores the calibration values for the I/O module and the probe from the requested UUT or USERDISK.

Arguments:

uut_userdisk	USERDISK or UUT Default = "USERDISK"
name	USERDISK or UUT name Default = "" (current USERDISK or UUT)

Example 1:

```
! restore calibration values from the current UUT
restorecal from "UUT"
```

Example 2:

```
! restore calibration value from the UUT DEMO
restorecal from "UUT", name "DEMO"
```

Example 3:

```
! restore calibration value from USERDISK /DR1
restorecal from "USERDISK", name "/DR1"
```

Remarks:

This function is similar to the front panel RESTORE CALDATA operation and restores calibration values from a TL/1 program. Calibration values may be restored from a USERDISK or UUT.

If the name of the USERDISK or UUT is the null string (""), then the current USERDISK or UUT is used. If the USERDISK or UUT is named, the calibration values are restored from the named USERDISK or UUT. Calibration values are restored for all I/O modules and the probe.

For More Information:

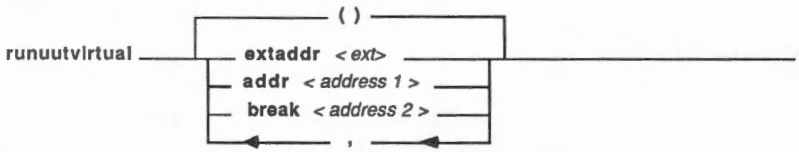
- The "Overview of TL/1" section of the *Programmer's Manual.*

runuutvirtual

Syntax:

```
runuutvirtual [extadddr <ext>,] addr <address 1>
 [, break <address 2>]

runuutvirtual (<address 1>, <address 2>)

runuutvirtual ()
```

Syntax Diagram:



Description:

Runuutvirtual is a complete replacement for the obsoleted *runuutspecial* command. *Runuutvirtual* causes the UUT to begin executing instructions from its own memory, asynchronously to the system. Execution begins at the virtual address specified. If none is specified, *runuutvirtual* defaults to an address that is pod dependent.

Arguments:

ext	Extended address bits used to form virtual addresses from address 1 and address 2.
address 1	Start address.
address 2	Breakpoint address. (Default = 0)

Example 1:

```
runuutvirtual ()
     ! start at pod-dependent starting
     ! address.
```

Example 2:

```
runuutvirtual extaddr 0, addr (read addr $FFFE)
     ! start at virtual address read from
     ! location FFFE
```

Example 3:

```
runuutvirtual extaddr 0, addr $1235, break $2000
     !  start at hex virtual address 1235
     !  stop at hex address 2000
```

Remarks:

Some pods have special addresses where a *runuut* may begin. Such addresses may read a reset or interrupt vector from memory and begin execution at that address.

Typically, *runuutvirtual* would be followed by *waituut* with a suitable maxtime value that allows the UUT time to complete its operation.

Some pods have a breakpoint capability which can optionally be enabled by specifying a stop address with the "break" argument. Generally, pods designed before the 80286-era cannot use the "break" feature. Refer to your pod manual for more specific information.

Execution of *runuutvirtual* continues until one of the following events occurs:

- The pod encounters a breakpoint.
- A DCE condition occurs.
- A *haltuut* is executed.
- The time specified in a *waituut* expires.
- The RESET key is press on the operator's keypad.
- The RUN UUT HALT command is entered from the operator's keypad.

Faults that occur during the execution of *runuutvirtual* are reported on the subsequent *haltuut* or *waituut*. Attempts to perform any pod-related operations except *waituut*, *haltuut*, or *polluut* results in an error if the *runuutvirtual* is still active. You must execute *haltuut* or *waituut* before attempting any other pod-related operations.

Related Commands:

compare, haltuut, polluut, waituut, runuut

For More Information:

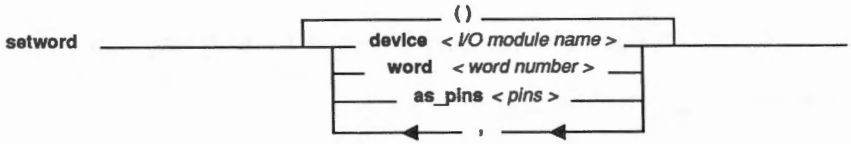
- The "Overview of TL/1" section of the *Programmer's Manual.*
- The Fluke pod manual for the microprocessor you are using.
- *Supplemental Pod Information for 9100A/9105A Users Manual.*

setword

Syntax:

```
setword [device <device name>] [, word <word number>]
     [, as_pins <pins>]

setword (<device name>, <word number> , <pins>)
```

Syntax Diagram:



Description:

Allows pin numbers to be grouped together to form a user defined word. *Setword* is identical to the front panel operation IOMOD SET WORD. Up to 40 unique pins may be grouped together for each I/O module in five different groups.

Arguments:

device name	I/O module name, ("/mod1", "/mod2", "/mod3", or "/mod4"). (Default = "/mod1")
word number	The number of the word group. Valid values are from 1 through 5. (Default = 1)
pins	A string of from 1 through 40 unique pin numbers. (Default = "40 39 38 37 ... 4 3 2 1")

Example 1:

```
setword device "/mod1", word 1, as_pins "1 2 3 4"
```

Example 2:

```
setword device "/mod1", word 5, as_pins "40 39 2 1"
```

Remarks:

The *setword* function is identical in operation to the front panel operation IOMOD SET WORD. The purpose of this command is to group together user related pins to form specific words for use with the *readword* and *writeword* commands.

Only unique pin numbers between 1 and 40 inclusive are valid.

Related Commands:

readword, writeword

For More Information:

- The "Overview of TL/1" section of the *Programmer's Manual.*

Syntax:

```
str num <expression 1> [, radix <expression 2>]

str (<expression 1>, <expression 2>)
```

Syntax Diagram:

Description:

Returns the string representation of the numeric operand.

Arguments:

expression 1	A numeric expression for the number to be converted into a string representation.
expression 2	A numeric expression for the radix of the number to be converted. The allowed radices are 2, 8, 10(default), or 16.

Returns:

The string representing the converted number.

Examples:

```
x = str(256,10) ! the variable x is set to the
                ! character string "256"

x = str(256,16) ! the variable x is set to the
                ! character string "100"
```

Related Commands:

fstr, val

Syntax:

```
val str <string> [, radix <radix>]

val (<string>, <radix>)
```

Syntax Diagram:

Description:

Calculates the numeric value of the string operand using the specified radix.

Arguments:

string	A string which represents a number.
radix	A numeric expression for the radix to use for the returned number. Allowed values for radix are 2, 8, 10 (default), and 16.

Returns:

A numeric value.

Example:

```
x = val ("15",16) ! the variable x is set to
                  ! the numeric value of
                  ! hexadecimal 15

x = val ("15",10) ! the variable x is set to
                  ! the numeric value of decimal 15
```

Related Commands:

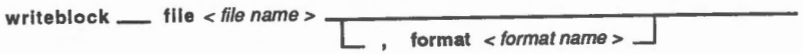
fval, str

writeblock

Syntax:

```
writeblock file <file name> [, format
     <format name>]

writeblock (<file name>, <format name>)
```

Syntax Diagram:



Description:

Loads the contents of a file in a standard ASCII form (Motorola S-Record format or Intel Hex format) into UUT or pod overlay RAM. The file contains information about the starting address and number of data bytes.

Arguments:

file name	The name of the file containing the required data.
format name	The ASCII format in which the data was previously stored. Either "motorola" or "intel". (Default = "motorola".)

Examples:

```
writeblock file "test_lcd", format "motorola"

writeblock ("pgm1", "motorola")
```

Related Commands:

readblock, loadblock

For More Information:

- The "Overview of TL/1" section of the *Programmer's Manual.*

writevirtual

Syntax:

```
writevirtual extaddr <ext>, addr <address>, data
      <data>

writevirtual (<ext>, <address>, <data>)
```

Syntax Diagram:

Description:

Writevirtual is a complete replacement for the obsoleted *writespecial* command. The *writevirtual* command writes the specified data to the specified virtual address. This allows access to the virtual addresses that, in some pods, are used for special operations. This command should only be used when you know that the normal *write* command does not provide the required special operation.

Arguments:

extaddr	Extended address bits.
address	The virtual address where data will be written.
data	Data to write.

Examples:

```
writevirtual extaddr 0, addr $F0000018, data $21
```

For More Information:

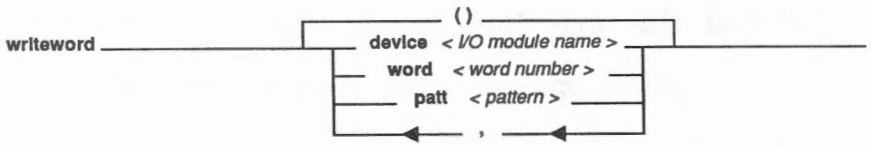
- The "Overview of TL/1" section of the *Programmer's Manual.*
- The Fluke pod manual for the microprocessor you are using.

writeword

Syntax:

```
writeword [device <I/O module name>] [ , word <word
     number>] [ , patt <pattern>]

writeword (<I/O module name> , <pattern> , <word number>)
```

Syntax Diagram:



Description:

Writes a data pattern to a group of I/O module pins. The group of I/O module pins is set using *setword.*

Arguments:

I/O module name	I/O module name ("/mod1", "/mod2", "/mod3", or "/mod4"). (Default = "/mod1")
word number	This specifies the pin grouping to use in writing out the word. (Default = 1)
pattern	Specifies the levels to be driven. Valid values are 1 (HIGH), 0 (LOW), and X or x (driver off, 3-stated) (Default = "0")

Example 1:

```
writeword device "/mod1", word 4, patt "10X"
```

Example 2:

```
writeword device "/mod2", word 1, patt "000000000000"
```

Remarks:

If not enough levels have been specified in the pattern, they are assumed to be LOW. If too many values are specified, the leading values are ignored.

Related Commands:

setword, readword

For More Information:

- The "Overview of TL/1" section of the *Programmer's Manual.*