Edit date: 07/27/83 G.Cossey

Lisa 1.75 Diagnostic tests.

This document contains a simple description of the diagnostics performed by the Lisa 1.75 Diagnostic packages.

ROM Checksum test.

These tests are done in the Boot ROM and are to verify that the correct data was programmed into each ROM and that each ROM location is readable. This test calculates the checksum of each ROM, using the same method as the Lisa 1.0 Boot ROM, and verifies that each ROM is correct.

1. For ROM #1. This ROM is the one that contains the Startup and Diagnostic code. If this ROM is bad then a Boot ROM fatal error will occur right away.
2. For ROM #2. This ROM contains hardware drivers and O.S. routines. If this ROM is bad then a fatal error will occur after all other diagnostics have been run.

Video Memory tests.

These tests assure that the screen area and the data area for the diagnostics are functioning properly. A failure here is fatal and will terminate testing.

1. Stuck-at tests, verifies that there are no stuck-at-one or stuck-at-zero memory locations.
2. Address uniqueness test, verifies that each memory location is unique and that no addresses are tied together.
3. Memory pattern tests, these tests look for other types of memory failures such as refresh fails.

Extensive. These tests are done in the Boot ROM. The Boot ROM uses the video memory as a data storage device and operation entirely from the CPU board, with no memory board plugged in, is essential for booting any device.

Burn-in and hard to find problems. This is a set of extended memory tests designed to be run from LisaTest. These tests run 'new and improved' pattern tests and are designed to catch very hard to find memory errors. These tests may run from only minutes, to hours, or days. The only way for these tests to be accessiable are for the Boot ROM's own tests to have passed.

Video circuitry tests.

This is a basic test of the video circuit. Both screen modes are timed to assure that the video is switching properly.

1. VRT/VID/CSYNC test.

The hardware provides a two bits in the status register which allows a program to monitor the video signals generated by the CPU board. This program tries to determine whether the CPU board video circuitry is working by looking at the VID and CSYNC releationships.

The test checks to see that there are transitions on the VRT (Vertical Retrace) bit in the status register, and makes a rough extimate of the time between pulses on this signal.

The CSYNC bit is tested for transitions and the VID bit is checked to make sure it is static for a solid black screen. The number of CSYNC pulses between vertical retraces is checked.

The third routine first writes an alternating pattern of 1s and 0s to the screen memory area, 45 words of 0 then 45 words of 1, etc., until the area is filled. This will create a pattern on the screen of alternate white and blank lines. The routine then uses a scan routine to review the status register between 2 vertical retrace pulses, and analyses the resulting data for the correct signal patterns.

After this, the program tests the vertical retrace interrupt capability.

Basic tests. These tests are designed for the Boot ROM and determine if the Video circuit is basically working.

Extensive timing tests. These tests are to be run from LisaTest and the LMO station. These do more extensive timing measurements of the video signals.

Serial number validity test.

This is a simple check to assure that the serial number is in the proper format and that the serial number is readable. A bounds check is done on parts of the serial number, this is a very loose test to allow all valid numbers to go thru. If the serial number is not readable or outside of legal bounds then a fatal Boot ROM error occurs.

This test is only required in the Boot ROM.

Timer tests.

The timer is one of the most important parts of the system. The various modes are tested for proper operation. A failure on either timer #0 or #1 will result in a fatal error for the Boot ROM. Since the timer #2 is used only for the speaker, its failure will result in a non-fatal error for the Boot ROM.

Timers #0, #1, and #2 will be tested (note: timer #2 does not generate interrupts).

1. Binary and BCD (Binary Coded Decimal) count modes.
2. Mode 0, Interrupt on terminal count.
3. Mode 1, Programmable One-Shot.
4. Mode 2, Rate Generator.
5. Mode 3, Square Wave Rate Generator.
6. Mode 4, Software Triggered Strobe.

Extensive tests. The Boot ROM will extensively test the timers for proper operation. These tests will test all of the modes above.

More extensive tests. More tests for the timers may be added to run thru LisaTest or LMO if the tests in the Boot ROM tests are not extensive enough.

COPS functional tests.

This is the device used to talk to the Keyboard and the Mouse. It also handles other items such as the time of day and programmable alarms.

1. Setable test.

The calendar control allows anyone to disable the calendar, which is done when it is being set. The first test is one to make sure that the calendar can be properly set. The calendar is disabled during this subtest, this assures that when it is set it will not change before the program has enough time to read it and see if it has been set properly.

The test sets the clock to many different values to assure that each part of the date is set to the correct values. The COPS is sent the date to be set and then the command is sent to the COPS to return the current date. Since the clock is disabled, the value set and the value returned will be the same.

2. Wraparound test.

This next test assures that the calendar keeps track of the time correctly. This assures that, for example, when the seconds go from 59 to 60 seconds that the calendar changes the seconds from 59 to 0 and increments the minutes by one. All the calendar parameters are tested to wraparound at the correct values. Other values are tested to assure that standard incrementing is done.

To do this test, the calendar is set to the test value, the calendar is enabled so it will increment every one tenth of a second. The program then waits for a little over one tenth of a second and then reads the current calendar time. Since the clock time should have changed, the time read will reflect the wraparound of parameters.

3. Alarm test.

The COPS device also contains a programmable alarm. The user can specify that an alarm be generated, in effect that the computer is interrupted by the COPS, after a specific amount of time. The resolution of the alarm is one second. The alarm is set, the program waits for the programmed time plus 3 seconds or until the interrupt occurs. The time expired from when the alarm was set and when the interrupt happened is verified against the actual programmed time.

4. Internal register tests.
Registers internal to the COPS are tested here.

5. Keyboard reset function.
The keyboard is reset by the COPS and the correct reset code is read. This value is compared to the reset code that was sent to the COPS on powerup.

Clock functional tests. These tests are for LisaTest and LMO stations. These do a complete functional check on the COPS and associated timing.

Keyboard reset code tests. These tests are simple functional tests that do not change any data (for example the date and time) and do only a basic check of the COPS. These tests are for all test areas, LisaTest and LMO and Boot ROM.

IWM diagnostic test.

This test trys to assure that the IWM chip and associated hardware on the CPU board is functional. It is a minimal test of this floppy interface since it is done whether or not a disk is in place and must assume that there is not a disk in place.

1. (To be determined)
2. (To be determined)
3. (To be determined)

Basic tests for IWM . These tests are tests of the IWM circuit and assume that the drive does not have a diskette installed, these tests are for CPU board LMO stations and for the Boot ROM.

Extensive tests for the IWM circuit going to the drive itself. These tests are for the IWM circuit and require that a drive and disk be installed, see the next test description.

Floppy disk tests.

These tests verify the proper operation of the floppy disk drive and complete the testing of the IWM and interface hardware. Since a floppy drive and disk are required to completely test the IWM circuit, these tests are required in the LMO station and LisaTest. Because of a Field Service requirement that the floppy drive be completely debugged from the Boot ROM, a subset of these tests will be in the Boot ROM as space permits.

1. (To be determined)
2. (To be determined)
3. (To be determined)

Basic tests for IWM and interface. These tests are tests of the IWM circuit and may have to assume that the drive is good, these tests are for CPU board LMO stations.

Extensive tests for drive itself. These tests are for the drive itself and may have to assume that the IWM circuit is good.

Note that there may be cases where both the IWM interface and the drive itself are unknowns. The tests must take this into consideration. But note that the best way to troubleshoot this case is to swap out the floppy drive (a simple procedure) and then re-run the test that has failed. This narrows the failure down to either the IWM circuit or the floppy. Once it is narrowed down then the tests for that specific circuit or device can be run to isolate the failure further.

RS232 tests.

These two serial ports are tested extensively for the most common modes that they will be used in. The ports own internal loopback feature helps to ensure a more complete test.

Port A and Port B.
- A) Register read/write tests.
- B) Local loopback tests.
  - a) Async test.
  - b) Sync test.
  - c) SDLC test.
- C) External loopback tests.
- D) Interrupt test.

A) Read/Write Test

This first module writes an $AA data pattern to an interval read/write register in Port A and then reads it back and verifies its integrity. It then repeats the process with a $55 pattern. Both patterns are then tested on port B. The main purpose of this test module is to verify that it is possible to access this device (i.e. it exists in valid address space). A special bus error trap is provided to retain control of the program in the event that the device is not readily accessible.

B) Internal Transmit/Receive Test

Here, the SCC device is tested for its ability to transmit and receive in asynchronous mode. The test is performed at 19200 baud using the internal loop (built in), therefore no external drivers are being tested as of yet. The data format is 8 bits, 1 stop bit, internal baud rate clock x16 mode, and odd parity. All possible 8 bit characters are used and each is checked for data integrity upon reception. Also after each character is transmitted the program waits for the "all sent" flag, data received flag, and then checks for special error conditions such as framing, parity, and overrun.

C) External Transmit/Receive Test

Here, the SCC device is tested for its ability to transmit and receive. The test is performed at all baud rates using the external loopback connector, therefore external drivers are being tested here. The data format is 8 bits, 1 stop bit, internal baud rate clock x16 mode, and odd parity. All possible 8 bit characters are used and each is checked for data integrity upon reception. Also after each character is transmitted the program waits for the "all sent" flag, data received flag, and then checks for special error conditions such as framing, parity, and overrun.

D) Interrupt Test.

Tests the ability of the SCC device to interrupt the system processor. First a dummy interrupt routine is setup to handle the interrupt. This routine resets the source of the interrupt in the SCC and restores the normal vector in the vector table. A default timeout is then set in the baud rate timer in the SCC and the device is programmed to interrupt the system processor on timeout. The system then begins a count and patiently awaits the expected interrupt.

Basic tests. These tests are the basics done to assure the ports work properly and will not drive any test data out the ports. These use the internal loopback feature and the latch that disables data going out the rear ports.

External tests (required loopback). These tests check the port in more of a 'natural' manner, in the way it is normally used.

MMU tests.

If a memory board is installed then this test is done to verify that the board can be mapped as the user requests.

1. Stuck-at tests, verifies that there are no stuck-at-one or stuck-at-zero memory locations.
2. Address uniqueness test, verifies that each memory location is unique and that no addresses are tied together.
3. Memory pattern tests, these tests look for other types of memory failures.

Basic pattern & functional tests. These tests are done by the Boot ROM and must pass for the boot process to continue. It may still be possible to do a auto-boot procedure on detailed diagnostics.

Extensive tests. These tests help to isolate failures to the chip. If space requirements prevent the Boot ROM diagnostics from isolating the failure far enough then these more extensive tests will do that task. These tests also do more extensive memory tests on the MMU RAMs.

Parity Circuit test.

This test verifies the memory parity check and generation circuits.

1. Parity detection test.

This program tests the capability of the CPU board to write wrong parity, (for diagnostic purposes), the Parity bit RAMs, and the failing memory address register. Parity is a method used in computers to assure that data stored in memory is correct. The common method of using parity is for the computer to count how many bits are on at each memory location and then add one if necessary, in the parity bit, to make it an odd number or leave the parity bit set to 0 if it already contains an odd number of bits on. Example:

Odd parity example:

Memory Data	Parity bit	# bits set
10001100000	0	3
10001110011	1	7

Using the capability of the CPU board to write wrong parity, the number of bits set above would be 4, parity bit set to 1, and 6, parity bit set to 0.

When the computer senses that a memory location being used contains the wrong parity bit, it assumes that the memory data has been changed in some way from the data that was written. When this error occurs the CPU board tells the computer about the error and also which memory location contains the error, this address is in the Failing Memory Address register.

This test uses the special circuit on the CPU board to set the wrong parity on some locations in memory, the parity detection circuit is disabled at this time. Parity detection is then enabled and the memory locations are read. Each location which has been set to the wrong parity is verified to generate a parity error and the failing address is checked.

2. This is a test of the parity memorys. These RAMs are tested in a special way since they must be 'written' and 'read' indirectly. These extended tests are very long because of the method that must be used.

3. Bus Timeout.
The last test is that for the bus timeout circuitry. If illegal memory, memory which is not in the system but still within legal address range, is accessed then a bus timeout will occur. The computer must recieve a signal back each time it addresses a location in memory. If the memory is not there then no signal is returned to the computer and the computer would sit and wait for the memory. To prevent the computer waiting forever, a timer is used to limit the length of time that the computer will wait.

Basic functional tests. These tests are in the Boot ROM and assure the basic operation of the functions described above.

Extensive tests. These are very extensive tests to determine parity circuit memory errors along with parity circuit errors.

Main Memory tests.

These tests perform a quick check of the memory. An optional extensive memory test is available thru the boot device code.

1. Stuck-at tests, verifies that there are no stuck-at-one or stuck-at-zero memory locations.

2. Address uniqueness test, verifies that each memory location is unique

and that no addresses are tied together.

3. Memory pattern tests, these tests look for other types of memory failures.

Basic tests. These tests reside in the Boot ROM and consist of tests in two classes. The first class are 'Stuck-at' and 'address' tests that are very fast and are run every time the system is booted up. The second class are 'memory pattern' tests that usually run a long time and are optional thru the Service mode.

Extensive tests. These are additional tests which run more 'exotic' patterns. These tests are loadable thru LisaTest or an Auto-boot diskette.

Additional MMU tests.

A functional test of the MMUs operation is done to assure that what goes into the MMU control, in the way of an address, actually has the desired effect. These are more extensive 'adder' tests.

1. MMU functional tests.

The MMU also can allow limited access to areas of memory. Memory can be specified as read only, so valuable information will not be destroyed, as write only, or as both read and writeable.

To convert a logical address, one which will work as described above, to a physical address, a real address that a memory board will recognize. A series of adders are used as shown below:

```
   Logical address bits
<--Segment Address--><---Segment Offset--> <-------Displacement--->
 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
```

The Segment Address addresses a register which contains a physical (real) address, this is called a Base address. This physical address contains valid bits in the bit 9 to bit 20 range, this is enough to access 2 Meg bytes of memory. This value is then added to the Logical address bits 0 to bit 16. The resultant address is the corresponding physical (real) address. See the example below:

```
   Logical address bits
<--Segment Address---> <--- Segment Offset ----><---Displacement--->
 23 22 21 20 19 18 17  16 15 14 13 12 11 10 9  8 7 6 5 4 3 2 1
  x  x  x  x  x  x  x   x  x  x  x  x  x  x  x  x x x x x x x x
 \__________________/  \_____________________________________________/
     \                                              \
      \__ SOR register (Segment Base Register)       \
                     \                                |
                      \                               |
            ___________\___________________           |
            20 19 18 17  16 15 14 13 12 11 10 9       |
(Add         x  x  x  x   x  x  x  x  x  x  x  x      |
these                   x x  x x  x x  x x  x x x x x x x x

(result is the physical address
                x x x x   x x  x x  x x  x x  x x x x x x x x
```

There are adders for bits 9 thru 20, bits 17 to 20 have an adder because of the possibility of a carry bit (16) having to be added. There are three physical adders used, one for bits 9 to 12, one for bits 13 to 16, and one for bits 17 to 20.

1. Adder tests.

The first section of this test verifies that all three adders are working. Sequence in the standard test is:

A) Low Adder Test (bits 9-12 of the logical address).
B) Middle Adder Test (bits 13-16 of the logical address).
C) Upper Adder Test (bits 17-20 of the logical address).

2. Adder Carry Generation Test.

The next section of the test assures that an adder carry will propagate from one adder to the next.

A lower adder is setup with the base register value $F and then a location in page $1 is accessed. So in the address translation process, a carry is generated from the lower adder into the middle adder. This method is used for the middle adder to the upper adder test.

A Long Carry test is done to assure that when the lower 2 adders are at $FF and page $1 is accessed that a carry will propagate from the lower adder to the middle adder to the upper adder. Thus in the address translation process, $FF + $1 = $100, generating a carry into the upper adder.

The Flip Adder Carry test is designed to propagate a carry thru all the adder bits. The base register is set to value $AA and page $56 is accessed. Since $AA + $56 = $100, a carry is generated thru all bits. The base register is next set to value $55 and page $AB is accessed. A carry is generated into the upper adder by the addition of $55 and $AB (= $100).

3. The last test is for Access violation.

The MMU has the capability to limit access to specific areas of memory for only specific purposes. The following conditions are setup and the limit detection is verified for each one.

a) Read/Write main memory. Reading and writing to a memory location is alright.

b) Read Only main memory. Any writing to this section of memory will not be allowed, it will be flagged as an error.

c) Read/Write stack. Calling procedures which place information on the stack and remove information from the stack will work alright.

d) Read Only stack. Calling procedures which place information on the stack will not be allowed, an error will be flagged. Pulling information from the stack will be allowed.

Built-in Hard Disk port diagnostic.

This test is designed to be run in the Boot ROM if the hard disk is ready at boot time, or if the hard disk was selected as the boot device. Other tests are designed as described below.

Non-destructive tests. These tests are for use with the disk's internal diagnostics. The disk has the capability of returning the status of it's own internal diagnostics. The disk may also have other internal diagnostic capabilitys that can be used without destroying any data on the disk.

Extensive destructive tests. These disk tests are destructive in nature and would be used only after all information has been backed up off of a suspected bad disk. These may also be run on a new disk before the Office System is backed up onto it.

Expansion card diagnostics.

Each expansion card has the capability of having its own diagnostic in a ROM that is part of that card. If a diagnostic resides on the card then it will be run as part of the normal Boot process, by the Boot ROM. If the card is also bootable, then if the card fails it's own internal diagnostic test it is made into an illegal boot device by the Boot ROM and the user will not be allowed to boot from it.

Speaker voltage tests (test card).

Designed to run from LisaTest or from an LMO station. This test will be identical in function as that on Lisa 1.0 systems. It basically tests the volume control and frequency of the signal that is sent directly to the speaker. While it can not tell if the speaker is working, it can not find a broken wire from the board to the speaker or a torn speaker, it can tell if the circuit that generates the speaker voltages is correct.

MAC sound voltage tests (test card).

Designed to run from LisaTest or from an LMO station. These tests are really enhancements to the 'Speaker voltage tests' described above. This test verifies that there is proper control of the sound signal by doing 'real time' measurements of the speaker voltage.

Contrast voltage tests (test card).

Designed to run from LisaTest or from an LMO station. This test will be identical in function as that on Lisa 1.0 systems. This tests that the contrast voltage can be changed in a smooth manner from a dark screen to a very bright screen, within the required voltage specifications.

Mouse port tests (test card).
Designed to run from LisaTest or from an LMO station. This test will be identical in function as that on Lisa 1.0 systems. This test uses the I/O Port Test card to simulate a mouse. All features of the COPS directly related to the Mouse are tested here.

Keyboard port tests (test card).
Designed to run from LisaTest or from an LMO station. This test will be identical in function as that on Lisa 1.0 systems. This test uses the I/O Port Test card to simulate a keyboard. The COPS/keyboard interface is tested by sending the COPS a series of keyboard reset codes. This tests the timing of the keyboard interface and also verifies that the COPS can properly recieve serial key codes from the keyboard.

DMA functional tests (test card).
Designed to run from LisaTest or from an LMO station. This test will be identical in function as that on Lisa 1.0 systems. This test uses the famous 'DMA Test Card' to verify that the DMA circuit works properly in the system.

C.Itoh Printer test.
Designed to run from LisaTest. This test will be identical in function as that on Lisa 1.0 systems. This is a confidence test of the printer and verifies all printer functions, those currently used and all others.

Qume Printer test.
Designed to run from LisaTest. This test will be identical in function as that on Lisa 1.0 systems. This is a confidence test of the printer and verifies all printer functions, those currently used and all others.

Mouse interactive test.
Designed to run from LisaTest. This test will be identical in function as that on Lisa 1.0 systems. This test is designed to test the Mouse for slipping, bumps and dents in the mouse ball, and button bounce. It also checks movements and measures the amount of pulses that the COPS has recieved.

Keyboard interactive test.

Designed to run from LisaTest. This test will be identical in function as that on Lisa 1.0 systems. This test is designed to look for bad keys in the keyboard. These bad keys consist of key bounce, sticking keys, dead keys, and keys that give the wrong keycode back to the system.

Floppy disk controller card test.

No loopback required. This test is similar to the diagnostic that will be built-in to this card's boot ROM. It does a test of the card with the assumption that something is connected on the other end and that 'something' must not be damaged.

Loopback required. This test is really a complete test of the board. Loopback capability is required to do a complete test. This card is currently in design and test specifics have yet to be defined.

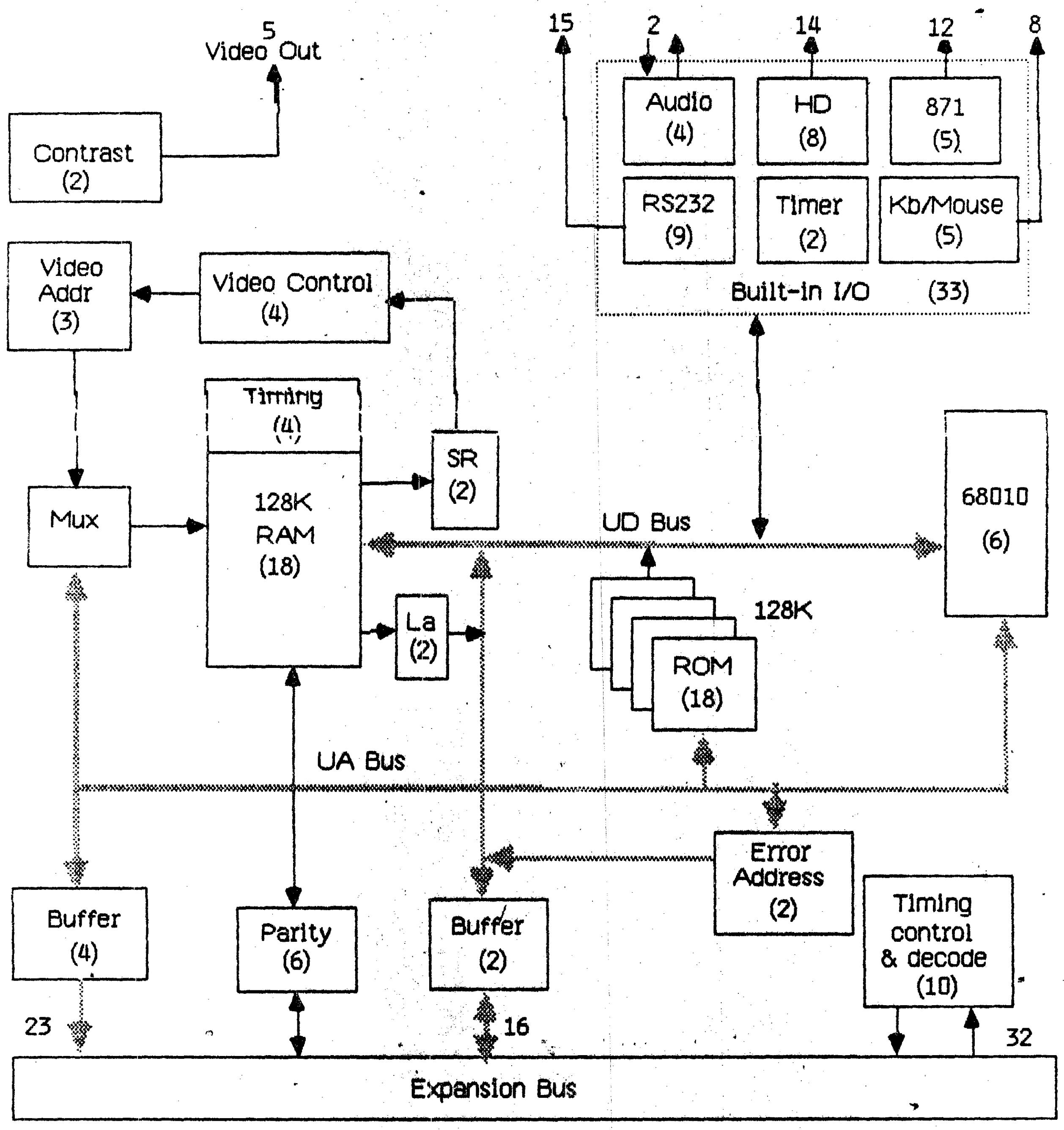
Two-port card test.

No loopback required. This test is similar to the diagnostic that will be builtin to this card's boot ROM. It does a test of the card with the assumption that something is connected on the other end and that 'something' must not be damaged. The two main devices, 6522s, are the main target of this test.

Loopback required. This test is really a complete test of the board. Loopback capability is required to do a complete test. What this test's main purpose is, is to check the port connections and do a continuity test of the lines from the 6522 devices out to the card's external connectors.

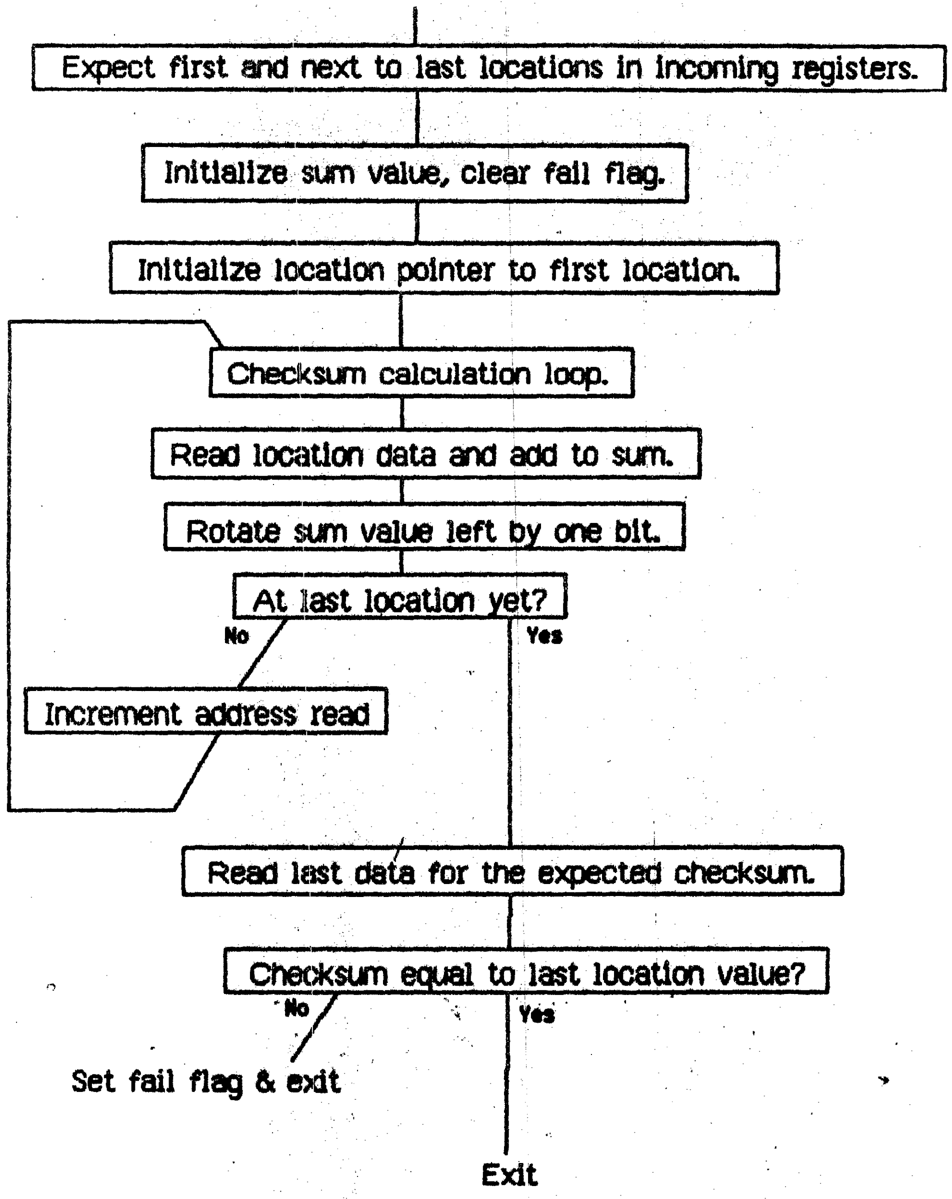
AppleNet card test.

No loopback required. To be defined.
Loopback required. To be defined.



LISA 1.75 CPU BOARD

Checksum calculations



Expect first and next to last locations in incoming registers.
Initialize sum value, clear fail flag.
Initialize location pointer to first location.
Checksum calculation loop.
Read location data and add to sum.
Rotate sum value left by one bit.
At last location yet?
No
Yes
Increment address read
Read last data for the expected checksum.
Checksum equal to last location value?
No
Yes
Set fail flag & exit
Exit

```
; Edit date: 08/03/83
;
;
; This program does a checksum test on any area of memory.
;
; Inputs:
;    a0 = First location
;    a1 = Last location
;
; Outputs:
;    d0 = Expected checksum
;    d1 = Actual checksum
;
;
;
CheckSum
        clr.l   d1              ;clear for checksum use
@1      add.w   (a0)+,d1        ;read location and add to sum
        rol     #1,d1           ;rotate to catch multiple bit errors
        cmpa.l  a0,a1           ;loop until done
        bne.s   @1
        move.w  (a0)+,d0        ;Expected checksum word
        rts
;
;
;
```

Timer #0 Test

Write mode register at $004007
Load counter #0 at $004001
Read counter #0 at $004001

16 bit counter mode
BCD counter mode

Load both bytes
Load only LSB byte
Load only MSB byte

Read both bytes
Read only LSB byte
Read only MSB byte

Initialize chip

All placed in mode 2.
All low bytes written to 0, disables counting & turns off interrupt request.
All placed in mode 0.

Basic register test.

LSB register test.

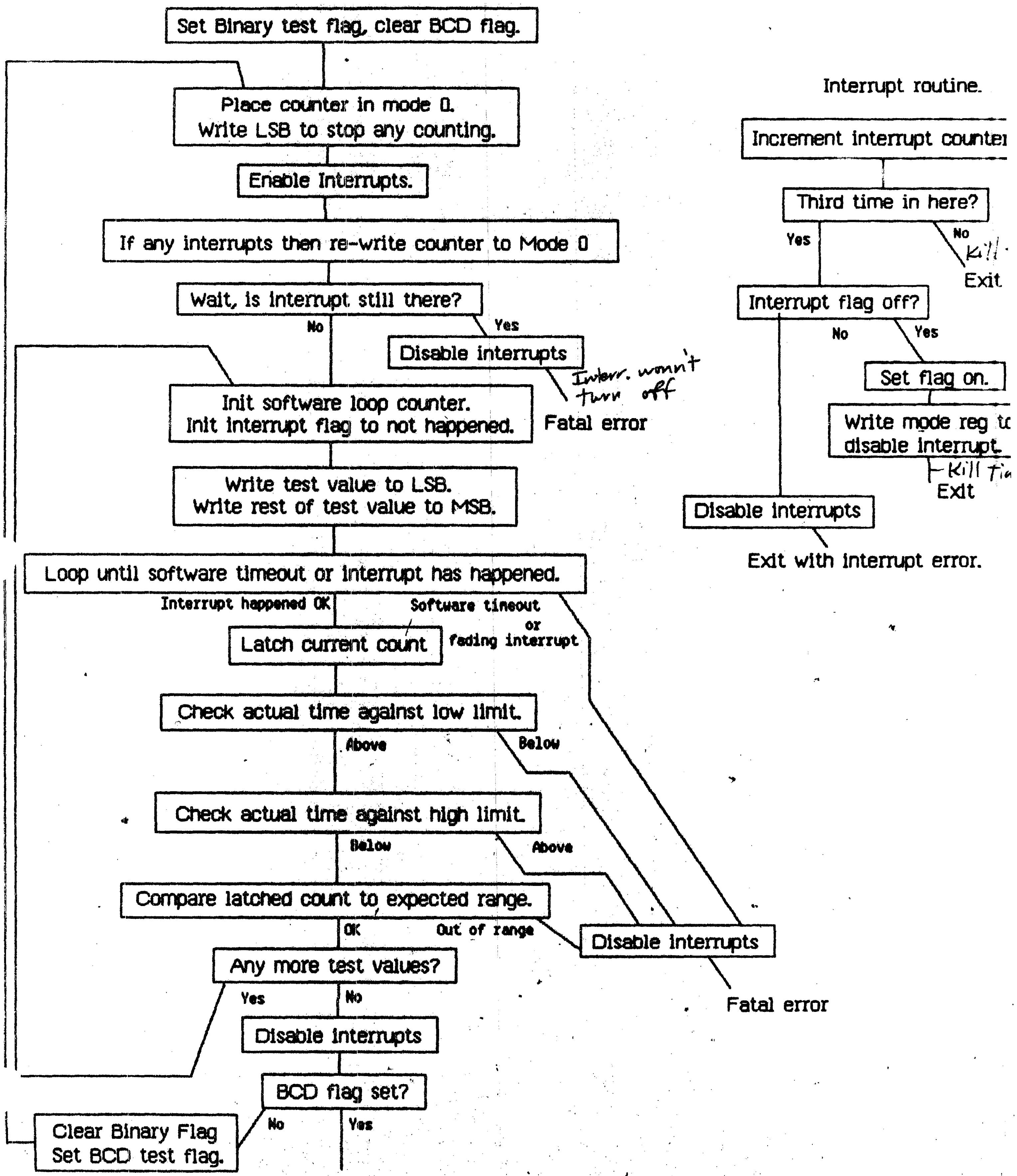
Write test value to LSB.
Read value from LSB & compare.
Abort test loop on failure.

MSB register test.

```
Place counter in mode 0.
    Write LSB to max value, allows time to do other steps before MSB decrements.
    Write MSB to test value.
    Latch counter value.
    Write LSB to stop counting.
    Read latched count and compare MSB to expected.
    Abort test loop on failure.
```

Mode 0 test.

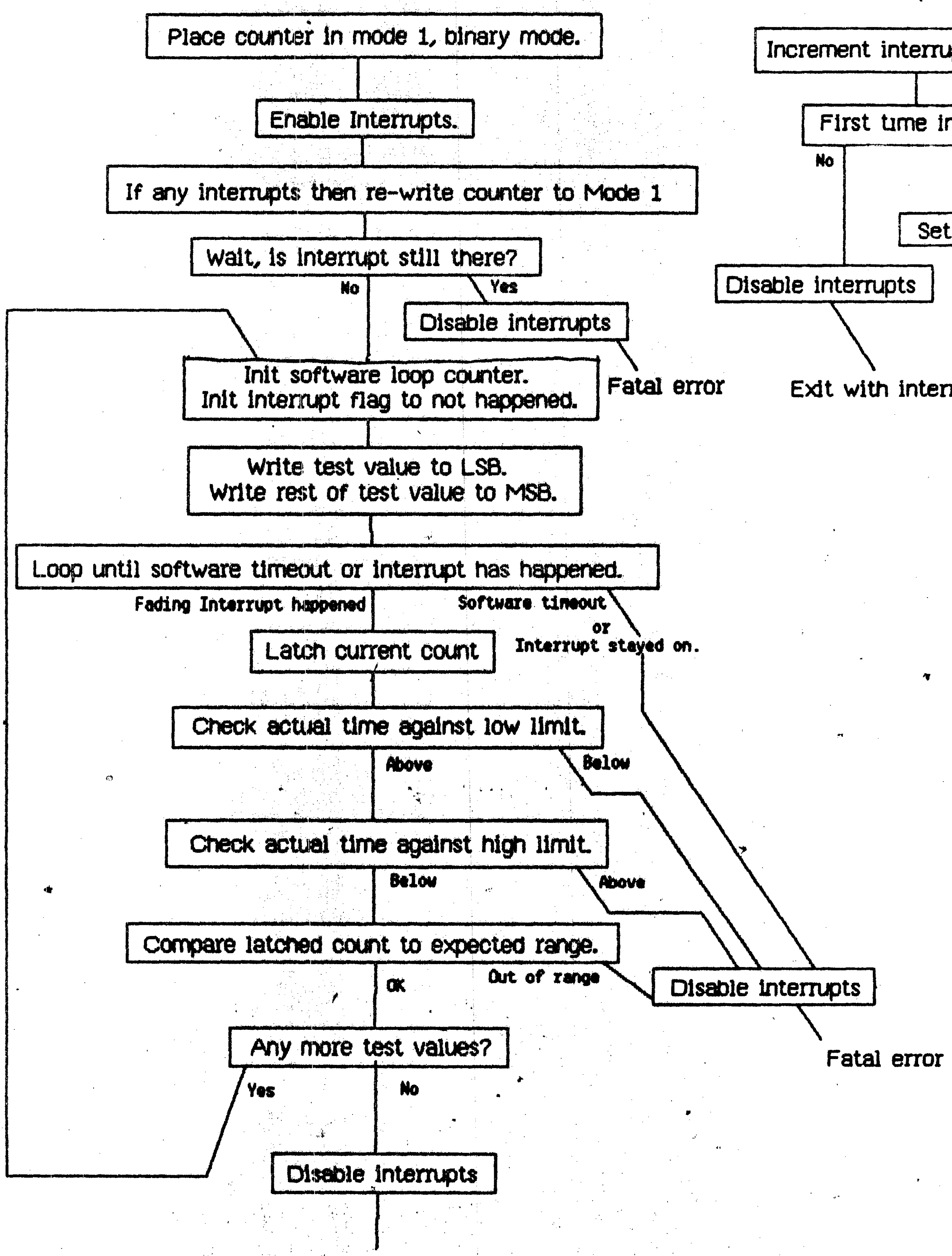
(0) Interrupt on terminal count mode.



Set Binary test flag, clear BCD flag.
Place counter in mode 0. Write LSB to stop any counting.
Enable Interrupts.
If any interrupts then re-write counter to Mode 0
Wait, is interrupt still there?
No
Yes
Disable interrupts
Interr. won't turn off
Fatal error
Init software loop counter. Init interrupt flag to not happened.
Write test value to LSB. Write rest of test value to MSB.
Loop until software timeout or interrupt has happened.
Interrupt happened OK
Software timeout or fading interrupt
Latch current count
Check actual time against low limit.
Above
Below
Check actual time against high limit.
Below
Above
Compare latched count to expected range.
OK
Out of range
Disable interrupts
Fatal error
Any more test values?
Yes
No
Disable interrupts
BCD flag set?
No
Yes
Clear Binary Flag Set BCD test flag.
Interrupt routine.
Increment interrupt counte
Third time in here?
Yes
No
Kill
Exit
Interrupt flag off?
No
Yes
Set flag on.
Write mode reg t disable interrupt.
Kill Ti
Exit
Disable interrupts
Exit with interrupt error.

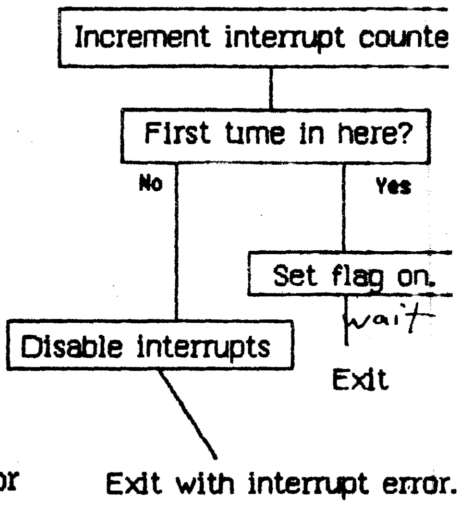
Mode 1 test.

(1) Programmable One-Shot mode.



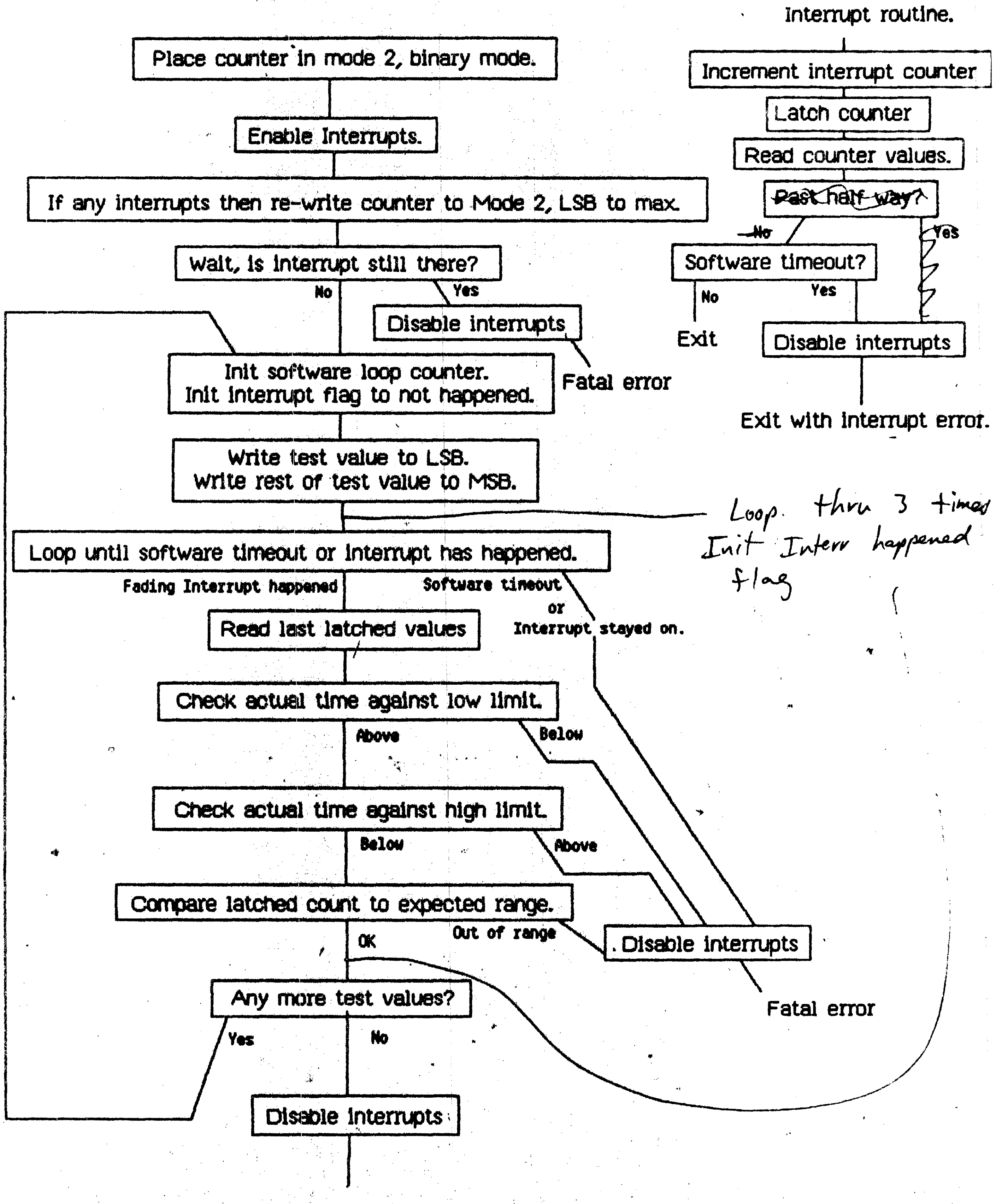
Place counter in mode 1, binary mode.
Enable Interrupts.
If any interrupts then re-write counter to Mode 1
Wait, is interrupt still there?
No
Yes
Disable interrupts
Fatal error
Init software loop counter.
Init interrupt flag to not happened.
Write test value to LSB.
Write rest of test value to MSB.
Loop until software timeout or interrupt has happened.
Fading Interrupt happened
Software timeout or Interrupt stayed on.
Latch current count
Check actual time against low limit.
Above
Below
Check actual time against high limit.
Below
Above
Compare latched count to expected range.
OK
Out of range
Disable interrupts
Fatal error
Any more test values?
Yes
No
Disable interrupts

Interrupt routine.



Mode 2 test.

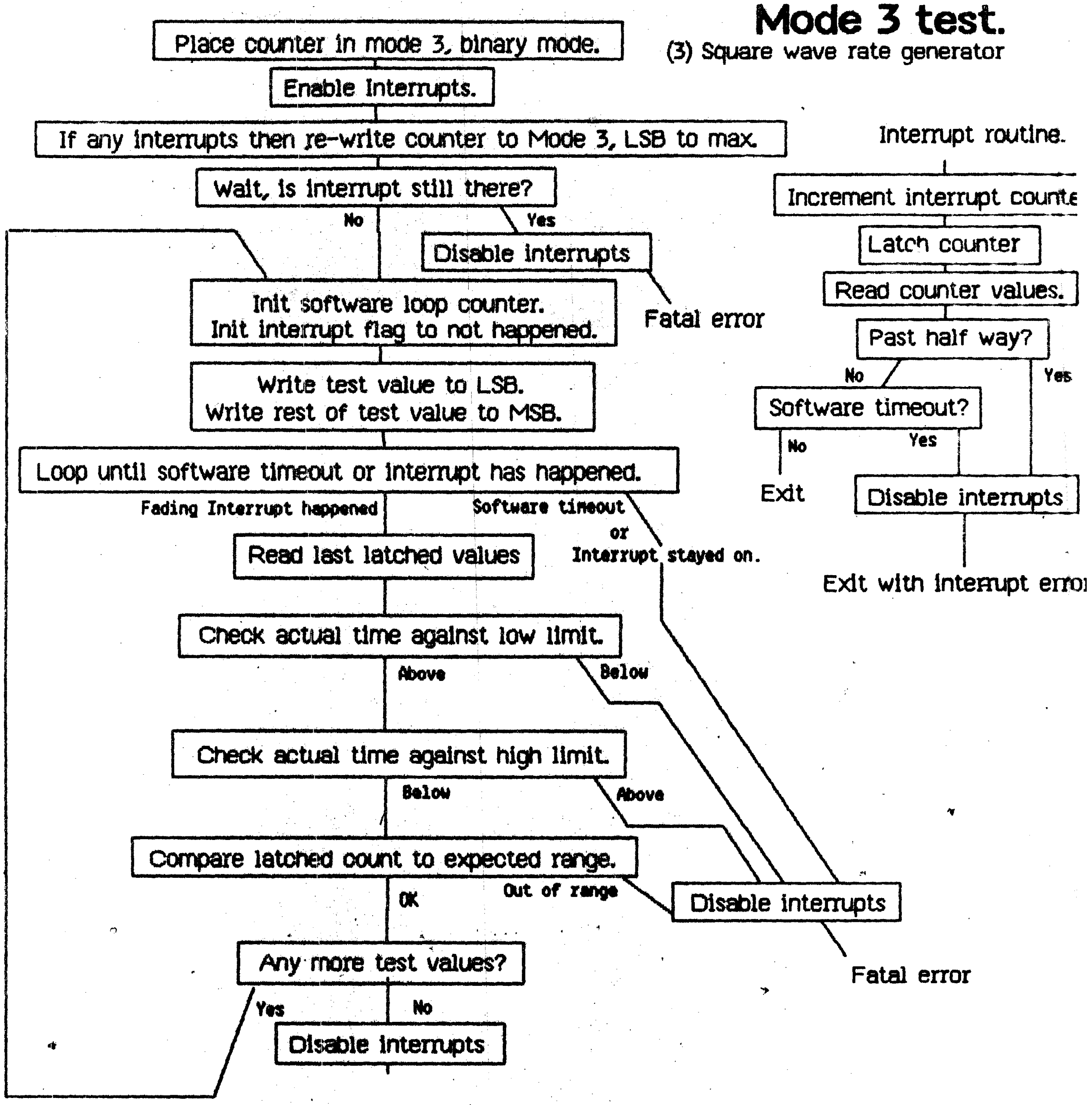
(2) Rate Generator mode.



Interrupt routine.
Increment interrupt counter
Latch counter
Read counter values.
~~Past half way~~?
~~No~~
Yes
Software timeout?
No
Yes
Exit
Disable interrupts
Exit with interrupt error.
Place counter in mode 2, binary mode.
Enable Interrupts.
If any interrupts then re-write counter to Mode 2, LSB to max.
Wait, is interrupt still there?
No
Yes
Disable interrupts
Fatal error
Init software loop counter.
Init interrupt flag to not happened.
Write test value to LSB.
Write rest of test value to MSB.
Loop thru 3 times
Init Interr happened flag
Loop until software timeout or interrupt has happened.
Fading Interrupt happened
Software timeout or Interrupt stayed on.
Read last latched values
Check actual time against low limit.
Above
Below
Check actual time against high limit.
Below
Above
Compare latched count to expected range.
OK
Out of range
Disable interrupts
Fatal error
Any more test values?
Yes
No
Disable interrupts

Mode 3 test.

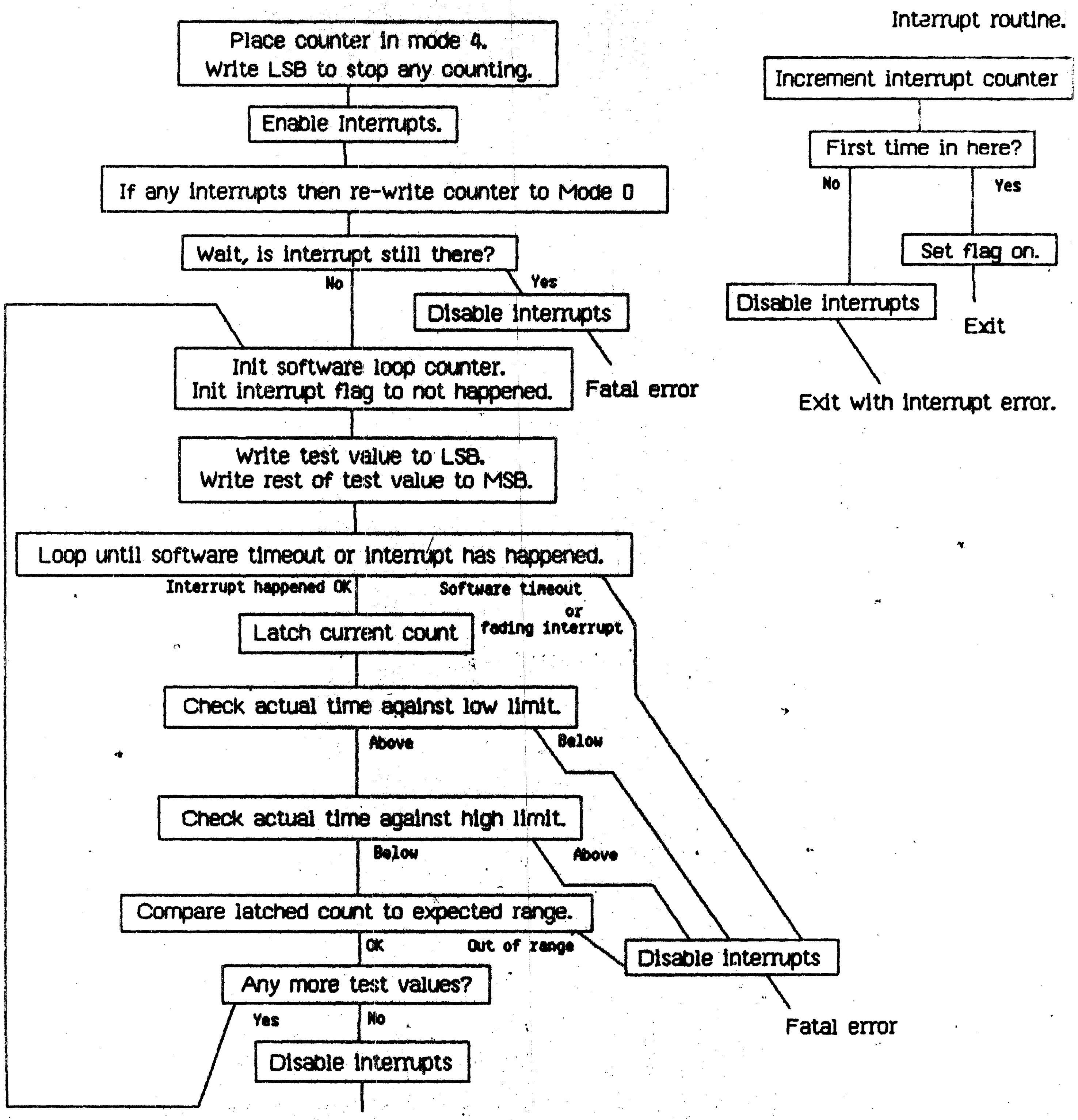
(3) Square wave rate generator



Place counter in mode 3, binary mode.
Enable Interrupts.
If any interrupts then re-write counter to Mode 3, LSB to max.
Wait, is interrupt still there?
No
Yes
Disable interrupts
Fatal error
Init software loop counter.
Init interrupt flag to not happened.
Write test value to LSB.
Write rest of test value to MSB.
Loop until software timeout or interrupt has happened.
Fading Interrupt happened
Software timeout or Interrupt stayed on.
Read last latched values
Check actual time against low limit.
Above
Below
Check actual time against high limit.
Below
Above
Compare latched count to expected range.
OK
Out of range
Disable interrupts
Fatal error
Any more test values?
Yes
No
Disable interrupts
Interrupt routine.
Increment interrupt counte
Latch counter
Read counter values.
Past half way?
No
Yes
Software timeout?
No
Yes
Exit
Disable interrupts
Exit with interrupt erro

Mode 4 test

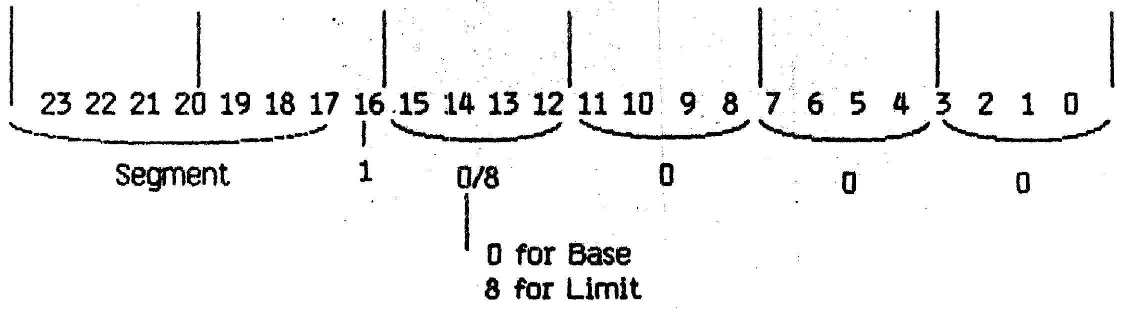
(4) Software Triggered Strobe



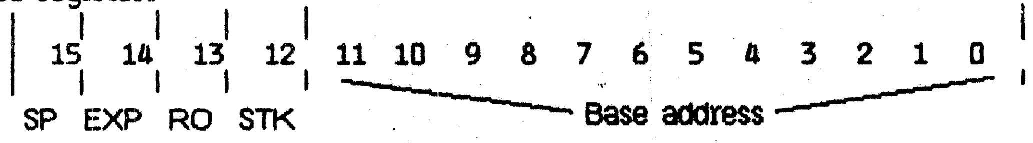
Place counter in mode 4. Write LSB to stop any counting.
Enable Interrupts.
If any interrupts then re-write counter to Mode 0
Wait, is interrupt still there?
No
Yes
Disable interrupts
Fatal error
Init software loop counter. Init interrupt flag to not happened.
Write test value to LSB. Write rest of test value to MSB.
Loop until software timeout or interrupt has happened.
Interrupt happened OK
Software timeout or fading interrupt
Latch current count
Check actual time against low limit.
Above
Below
Check actual time against high limit.
Below
Above
Compare latched count to expected range.
OK
Out of range
Disable interrupts
Fatal error
Any more test values?
Yes
No
Disable interrupts
Interrupt routine.
Increment interrupt counter
First time in here?
No
Yes
Set flag on.
Exit
Disable interrupts
Exit with interrupt error.

MMU Basics

Accessing MMU registers

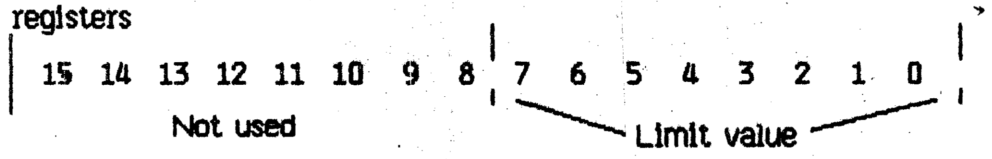


Base registers



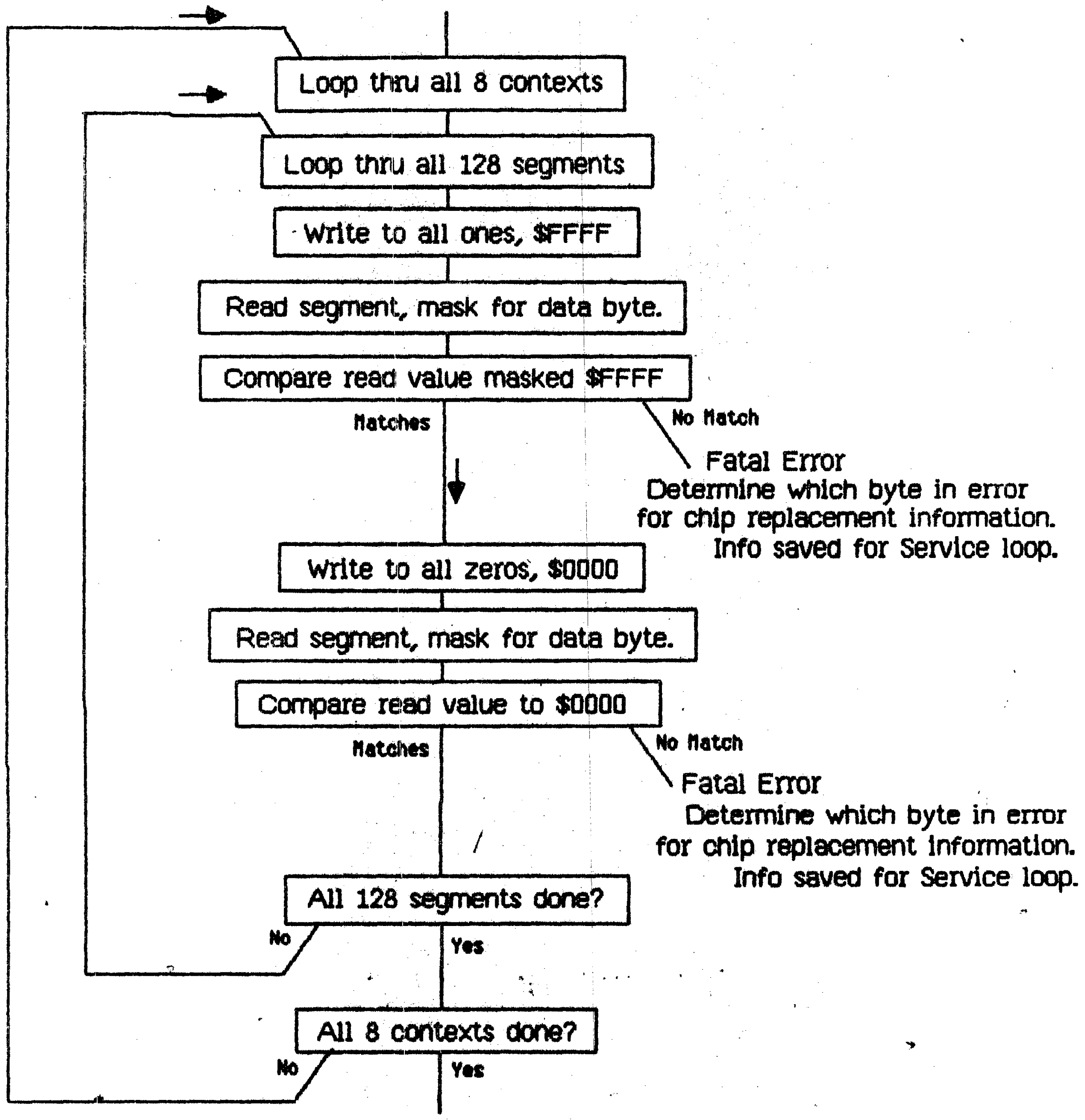
Data mask of $FFFF is used.

Limit registers

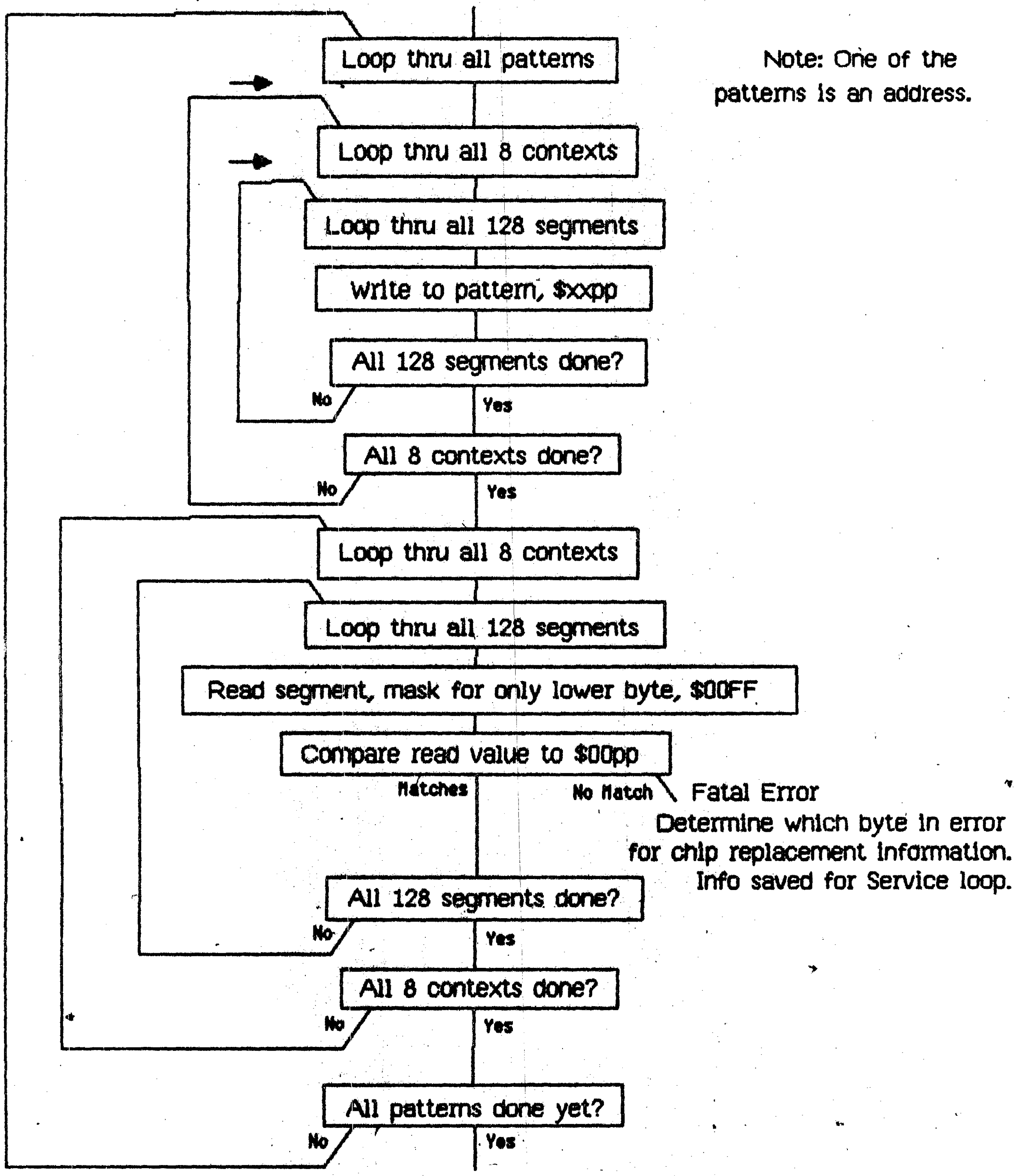


Data mask of $00FF is used.

Stuck-at tests.

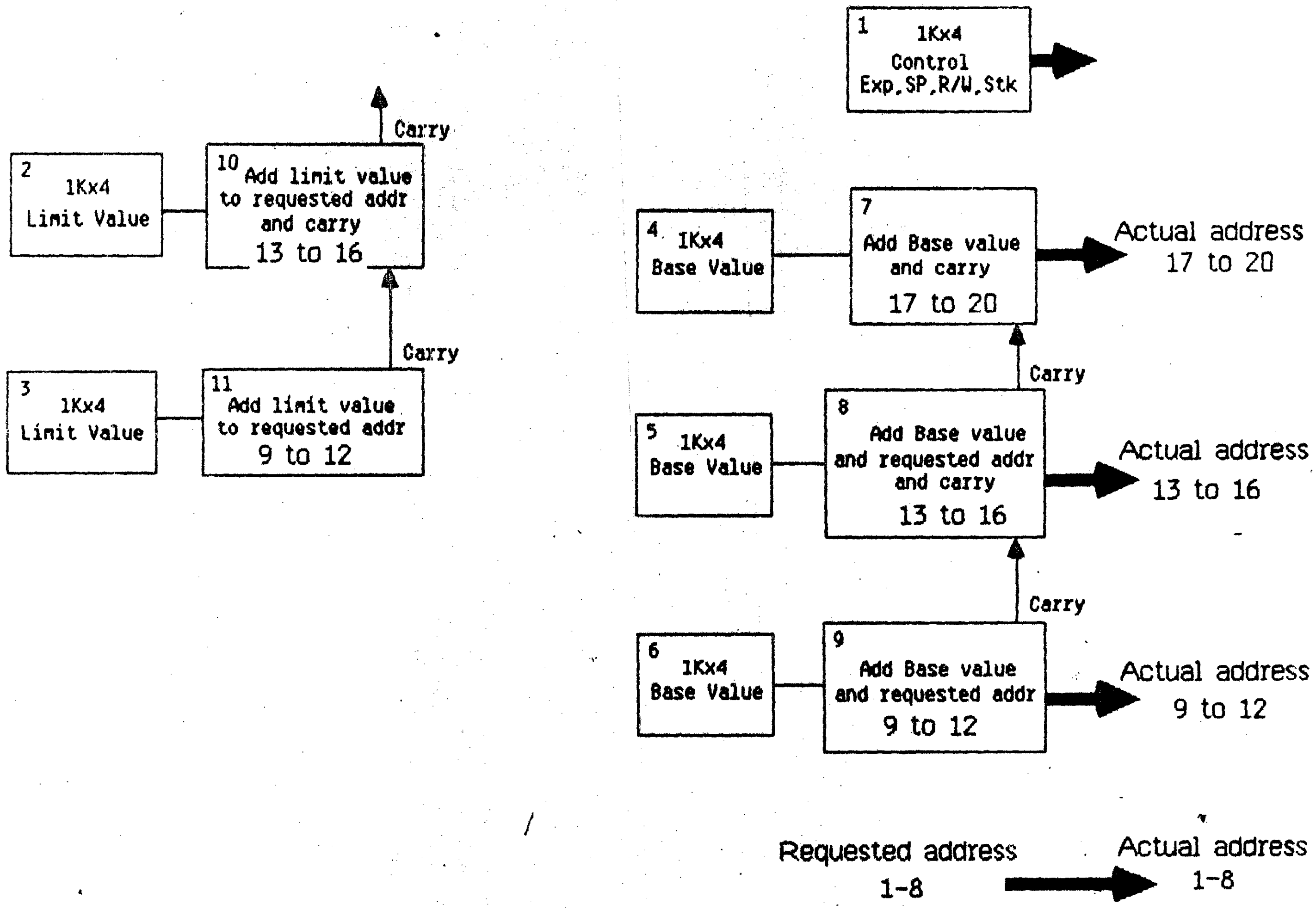

Loop thru all 8 contexts
Loop thru all 128 segments
Write to all ones, $FFFF
Read segment, mask for data byte.
Compare read value masked $FFFF
Matches
No Match
Fatal Error
Determine which byte in error
for chip replacement information.
Info saved for Service loop.
Write to all zeros, $0000
Read segment, mask for data byte.
Compare read value to $0000
Matches
No Match
Fatal Error
Determine which byte in error
for chip replacement information.
Info saved for Service loop.
All 128 segments done?
No
Yes
All 8 contexts done?
No
Yes

Read/Write tests.



Loop thru all patterns
Note: One of the patterns is an address.
Loop thru all 8 contexts
Loop thru all 128 segments
Write to pattern, $xxpp
All 128 segments done?
No
Yes
All 8 contexts done?
No
Yes
Loop thru all 8 contexts
Loop thru all 128 segments
Read segment, mask for only lower byte, $00FF
Compare read value to $00pp
Matches
No Match
Fatal Error
Determine which byte in error
for chip replacement information.
Info saved for Service loop.
All 128 segments done?
No
Yes
All 8 contexts done?
No
Yes
All patterns done yet?
No
Yes

MMU Structure



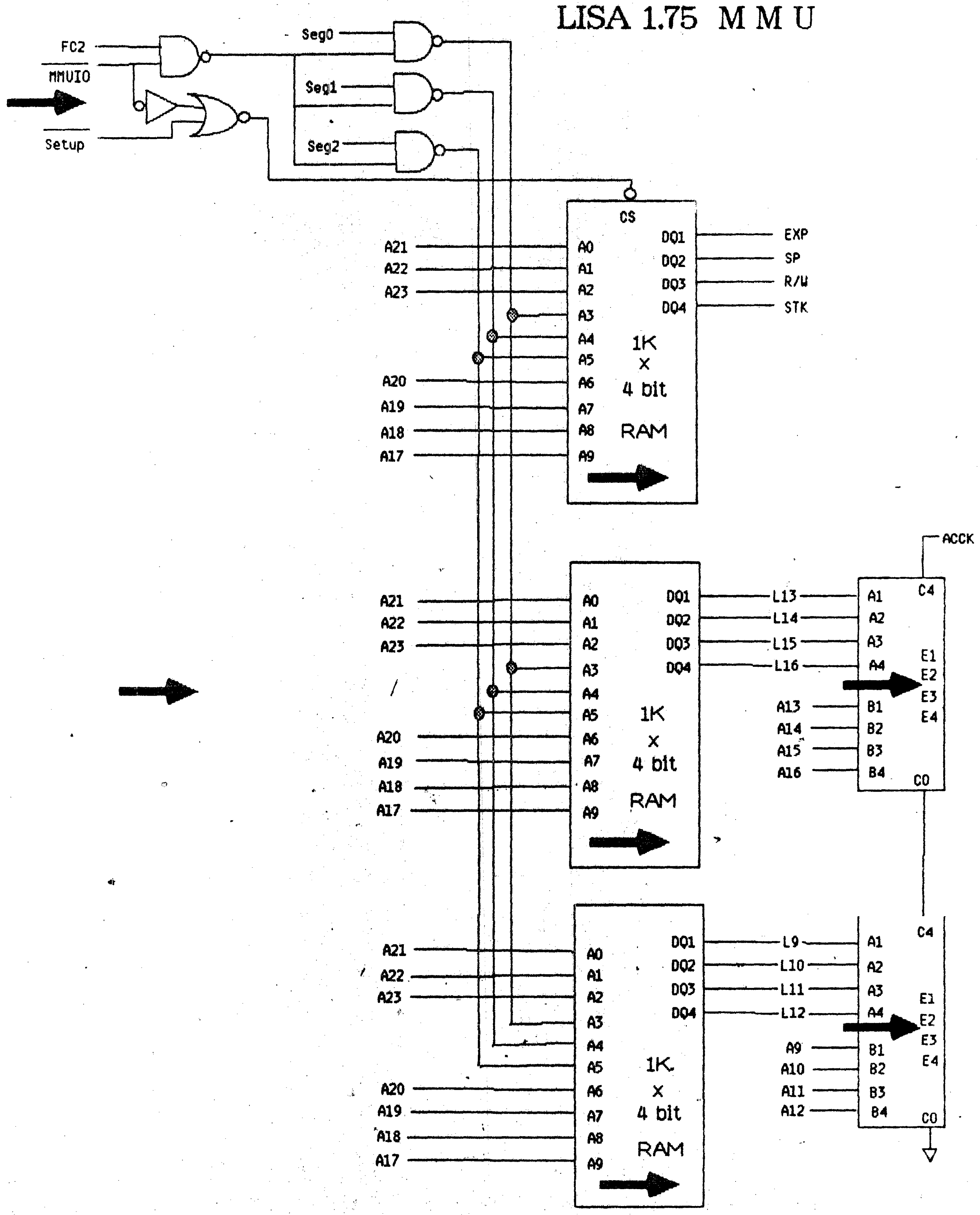
1 1Kx4 Control Exp,SP,R/W,Stk
2 1Kx4 Limit Value
10 Add limit value to requested addr and carry 13 to 16
Carry
3 1Kx4 Limit Value
11 Add limit value to requested addr 9 to 12
Carry
4 1Kx4 Base Value
7 Add Base value and carry 17 to 20
Actual address 17 to 20
Carry
5 1Kx4 Base Value
8 Add Base value and requested addr and carry 13 to 16
Actual address 13 to 16
Carry
6 1Kx4 Base Value
9 Add Base value and requested addr 9 to 12
Actual address 9 to 12
Requested address 1-8
Actual address 1-8

1 to 6 tested by Ram tests.
Failure is either 1Kx4 RAM, LS245 transceiver, or LS373 xxxx.
Displayed error will show RAM, LS245, and then LS373 (in that order).

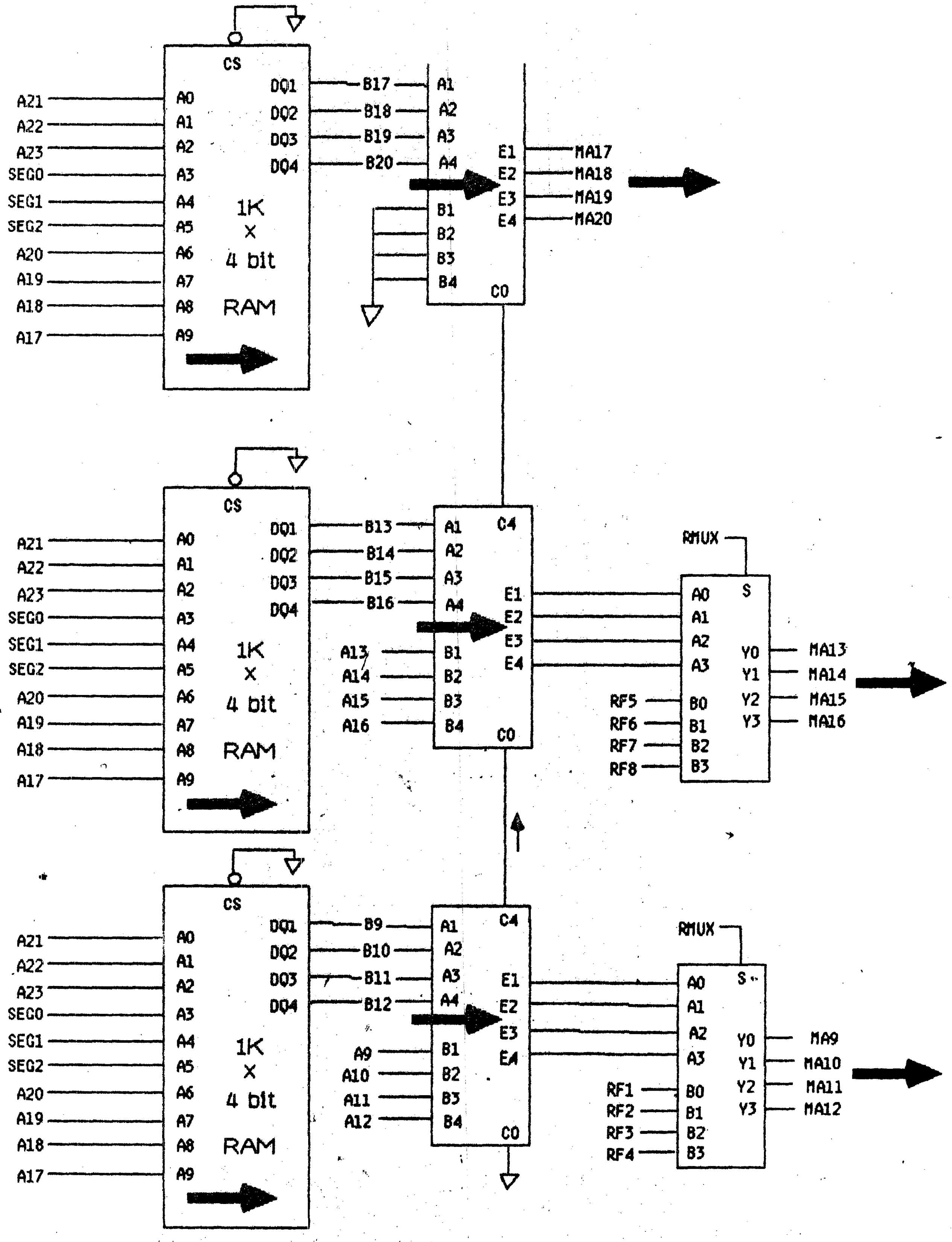
7 to 9 tested by functional tests, addresses actually accessed must agree with those expected.
The failure will display the 283 Adder.

10 to 11 by functional tests, limit failures and legal values are both used to determine if the adders are working correctly.
The failure will display the 283 Adder.

LISA 1.75 M M U



FC2
MMUIO
Setup
Seg0
Seg1
Seg2
CS
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
DQ1
DQ2
DQ3
DQ4
EXP
SP
R/W
STK
1K x 4 bit RAM
A21
A22
A23
A20
A19
A18
A17
L13
L14
L15
L16
L9
L10
L11
L12
A13
A14
A15
A16
A9
A10
A11
A12
B1
B2
B3
B4
C4
E1
E2
E3
E4
CO
ACCK



CS
A21
A22
A23
SEG0
SEG1
SEG2
A20
A19
A18
A17
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
DQ1
DQ2
DQ3
DQ4
1K x 4 bit RAM
B17
B18
B19
B20
B1
B2
B3
B4
E1
E2
E3
E4
C0
C4
MA17
MA18
MA19
MA20
B13
B14
B15
B16
A13
A14
A15
A16
RMUX
S
Y0
Y1
Y2
Y3
RF5
RF6
RF7
RF8
MA13
MA14
MA15
MA16
B9
B10
B11
B12
A9
A10
A11
A12
RF1
RF2
RF3
RF4
MA9
MA10
MA11
MA12

```
;
;  Edit date: 08/04/83
;
;
;==========================================================================;
;
;  MMU REGISTER TEST PROGRAM
;
;  FILE NAMES:
;       MMURAM.HDR.TEXT
;       MMURAM.ASM.TEXT
;
;  FUNCTION:
;
;       This is a test of the MMU RAMs on the memory board for Lisa 1.75
;
;       This test is expected to run from either ROM or from LisaTest.  The ROM
;       version is a conditional assembly subset of the LisaTest version.
;
;       Tests MMU registers by doing read/write, address, and a moving
;       inversions test. Errors are collected and written to memory for
;       later inspection.
;
;       After initialization, any traps will be caught and all registers
;       are saved for perusal by user.  However, during the testing when
;       the SETUP bit is on, any exceptions will cause the exception vector
;       to be fetched from the BOOT ROM and the BOOT ROM exception routine
;       will be executed.
;
;
;  LOGIC TESTED:
;
;       The following logic on the CPU board is tested:
;
;       MMU registers for contexts 0, 1, 2, and  3
;
;       The following MMU logic is not tested:
;
;       Other MMU adder and limit checking logic
;       MMU context 0 segments 0, 1, 126, and 127 registers
;
;
;  TECH OPTIONS:
;       In addition to the three standard tech-mode options, this program
;       supplies the following:
;
;       $0010 (16):
;
;
;  STEP:
;       $01:
;
;  ERROR CODES:
;       $00:  no error
;
;   HARDWARE:
;       CPU board.
;
;   THEORY OF TEST OPERATION:
;
;
;
;
;   CAUSES OF ERRORS
;
;       Most problems caught by this test will be RAM bit errors; gross
;       problems will be caught before this test is ever run by the BOOT
;       RAM tests; occasionally a problem may cause the program to hang
;       or go back to the ROM - note that any exceptions which occur during testing
;       with the SETUP bit on will cause the 68000 to fetch the exception
;       vector from the BOOT ROM.
;
;
;
;  ORIGINATOR:  George Cossey 08/04/83
;
;  MODIFIED BY:
;
;   TO BE DONE:
;
;
;=======================================================================
;
; Register usage:
;
;   d0 - Context (0 to 7)                 a0 - Base address being tested
;   d1 - Segment in context (0 to 127)    a1 - Limit address being tested
;   d2 - Write data (lower & upper)       a2 -
;   d3 - Read data  (lower & upper)       a3 -
;   d4 -                                  a4 - Parameter address, passed
;   d5 -                                  a5 - Scratch
;   d6 - Scratch                          a6 - Scratch
;   d7 - Scratch                          a7 - Stack
;
;=======================================================================
;
```

```
        .PROC   MMUREG,0
        .ORG    $0800
;
;
BSTART
;  Save video memory area.  (LisaTest version only)
     bsr     SAVEVID
;
;  Move test into video area.  (LisaTest version only)
     bsr     MOVETOVID
     jmp     (a4)              ;Jump to VIDEOSTART in video memory
;
;==================================================================
;
VIDEOSTART
;
     tst     $00003E00         ;Turn on SETUP bit
;
;  Save all of MMU RAM data.  (LisaTest version only)
     lea     BMMUSAVE,a4       ;Location to save MMU information
     bsr     FULLMMU
;
     lea     BASE4,a5          ;Clear failed RAM flags
     clr.l   d6
     move.l  d6,(a5)+
     move.l  d6,(a5)+
     move.l  d6,(a5)+
;
;  Pattern tests.
;     a) All zeros (0's).
     clr.l   d2
     bsr     PATTERN
;
;     b) All ones  (1's).
     move.l  #$ffffffff,d2
     bsr     PATTERN
;
;     c) Checkerboard.
     move.l  #$AAAA5555,d2
     bsr     PATTERN
;
;     d) Inverse checkerboard.
     move.l  #$5555AAAA,d2
     bsr     PATTERN
;
;     e) Address.
     bsr     ADDRESS
;
;     f) Moving Inversions.
     bsr     INVERSIONS
;
;
;  Restore all of MMU RAM data.  (LisaTest version only)
     lea     BMMUSAVE,a4       ;Location to save MMU information
     bsr     RESTOREMMU
;
;  Move back to main memory.  (LisaTest version only)
     jmp     JUMPBACK
;
;==================================================================
;
JUMPBACK
     tst     $00003F00         ;Turn off SETUP bit
;
;  Restore video memory area.  (LisaTest version only)
     bsr     VIDRESTORE
;
     rts
;
;==========================================================================
;==========================================================================
;
; Save video memory before test is moved up there
;
SAVEVID
     lea     BSTART,a3         ;Start of test
     lea     BEND,a4           ;end of test
     lea     BSAVE,a5          ;Area to save test in
     move.l  $00000800,a6      ;Video memory test area
;
@1   move.l  (a6)+,(a5)+       ;Move from video to save area
     adda    #4,a3             ;keep track of size
     cmp.l   a4,a3             ;done all of test?
     bmi     @1                ;...no
     rts
;___________________________________________________________________________
;
; Move test to video memory, exits with transfer address in A4
;
MOVETOVID
     lea     BEND,a4
     lea     BSTART,a5         ;Start of test
     move.l  $00000800,a6      ;Video memory test area
;
@1   move.l  (a5)+,(a6)+       ;Move from main memory to video memory
     cmp.l   a4,a5             ;done all of test?
```

```
        bmi     @1                      ;...no
;
        lea     VIDEOSTART,a4           ;Location of where to branch to
        lea     BSTART,a5               ;Start of test
        suba    a5,a4                   ;(VIDEOSTART - BSTART)
        adda    #$00000800,a4           ;+ (Start in video memory)
        rts
;-----------------------------------------------------------------------
;
;  Save all of MMU RAM data to area pointed to by A4
;
FULLMMU
; Set context, 0 to 7
        clr.w   d0                      ;Context
@1      bsr     SetContext
        clr.w   d1                      ;Segment
@2      bsr     SetSegment
        move.b  (a1),(a4)+              ;Read and save Limit register value
        move.w  (a0),(a4)+              ;Read and save Base register value
        add.w   #1,d1                   ;Next segment
        cmp.w   #128,d1                 ;..done all?
        bne     @2
        add.w   #1,d0                   ;Next context
        cmp.w   #8,d0                   ;..done all?
        bne     @1
        rts
;-----------------------------------------------------------------------
;
;  Restore all of MMU RAM data from area at A4 (in FULLMMU saved format)
RESTOREMMU
; Set context, 0 to 7
        clr.w   d0                      ;Context
@1      bsr     SetContext
        clr.w   d1                      ;Segment
@2      bsr     SetSegment
        move.b  (a1)+,(a3)              ;Restore Limit register value
        move.w  (a0)+,(a5)              ;Restore Base register value
        add.w   #1,d1                   ;Next segment
        cmp.w   #128,d1                 ;..done all?
        bne     @2
        add.w   #1,d0                   ;Next context
        cmp.w   #8,d0                   ;..done all?
        bne     @1
        rts
;-----------------------------------------------------------------------
;
; Save video memory before test is moved up there
;
VIDRESTORE
        lea     BSTART,a3               ;Start of test
        lea     BEND,a4                 ;end of test
        lea     BSAVE,a5                ;Area to restore from
        move.l  $00000800,a6            ;Video memory test area
;
@1      move.l  (a5)+,(a6)+             ;Move from save/area to video
        adda    #4,a3                   ;keep track of size
        cmp.l   a4,a3                   ;done all of test?
        bmi     @1                      ;...no
        rts
;-----------------------------------------------------------------------
;
; Pattern to write comes in inside of word d2.
; This pattern is written alternating to each ram location.
;
PATTERN
;
; Loop thru all contexts and segments and write test pattern
        bsr     WPAT
;
;        Loop thru all contexts and segments, read and compare data.
        clr.w   d0                      ;Context
@1      bsr     SetContext
        clr.w   d1                      ;Segment
@2      bsr     SetSegment
        move.b  (a1),d3                 ;Read Limit register
        cmp.b   d2,d3                   ;...same as written?
        beq     @3                      ;...yes, continue
        bsr     BADLIMIT                ;...no, flag failure
;
@3      move.w  (a0),d3                 ;Read Base register
        cmp.w   d2,d3                   ;...same as written?
        beq     @4                      ;...yes, continue
        bsr     BADBASE                 ;...no, flag failure
;
@4      add.w   #1,d1                   ;Next segment
        swap    d2
        bsr     SetSegment
        move.b  (a1),d3                 ;Read Limit register
        cmp.b   d2,d3                   ;...same as written?
        beq     @5                      ;...yes, continue
        bsr     BADLIMIT                ;...no, flag failure
;
@5      move.w  (a0),d3                 ;Read Base register
        cmp.w   d2,d3                   ;...same as written?
        beq     @6                      ;...yes, continue
        bsr     BADBASE                 ;...no, flag failure
```

```
;
@6    add.w   #1,d1             ;Next segment
      cmp.w   #128,d1           ;..done all?
      bne     @2
      add.w   #1,d0             ;Next context
      cmp.w   #8,d0             ;..done all?
      bne     @1
      rts
;
;______________________________________________________________
;
; Write a pattern
;
WPAT
      clr.w   d0                ;Context
@1    bsr     SetContext
      clr.w   d1                ;Segment
@2    bsr     SetSegment
      move.b  d2,(a1)           ;Write to Limit register
      move.w  d2,(a0)           ;Write to Base register
      add.w   #1,d1             ;Next segment
      swap    d2
      bsr     SetSegment
      move.b  d2,(a1)           ;Write to Limit register
      move.w  d2,(a0)           ;Write to Base register
      swap    d2
      add.w   #1,d1             ;Next segment
      cmp.w   #128,d1           ;..done all?
      bne     @2
      add.w   #1,d0             ;Next context
      cmp.w   #8,d0             ;..done all?
      bne     @1
      rts
;______________________________________________________________
;
; Address test.
;
ADDRESS
;
; 1.  Write addresses from low to high
;     Read all registers and verify
;
; Loop thru all contexts and segments and write test pattern
      clr.l   d2                ;Address pattern
      clr.w   d0                ;Context
@1    bsr     SetContext
      clr.w   d1                ;Segment
@2    bsr     SetSegment
      move.b  d2,(a1)           ;Write to Limit register
      add.w   #$0101,d2         ;Next address
      move.w  d2,(a0)           ;Write to Base register
      add.w   #$0101,d2         ;Next address
      add.w   #1,d1             ;Next segment
      cmp.w   #128,d1           ;..done all?
      bne     @2
      add.w   #1,d0             ;Next context
      cmp.w   #8,d0             ;..done all?
      bne     @1
;
;
;     Loop thru all contexts and segments, read and compare data.
      clr.l   d2                ;Address pattern
      clr.w   d0                ;Context
@3    bsr     SetContext
      clr.w   d1                ;Segment
@4    bsr     SetSegment
      move.b  (a1),d3           ;Read Limit register
      cmp.b   d2,d3             ;...same as written?
      beq     @5                ;...yes, continue
      bsr     BADLIMIT          ;...no, flag failure
;
@5    add.w   #$0101,d2         ;Next address
      move.w  (a0),d3           ;Read Base register
      cmp.w   d2,d3             ;...same as written?
      beq     @6                ;...yes, continue
      bsr     BADBASE           ;...no, flag failure
;
@6    add.w   #$0101,d2         ;Next address
      add.w   #1,d1             ;Next segment
      cmp.w   #128,d1           ;..done all?
      bne     @2
      add.w   #1,d0             ;Next context
      cmp.w   #8,d0             ;..done all?
      bne     @1
;
;
; 2.  Write addresses from high to low
;     Read all registers and verify
;
; Loop thru all contexts and segments and write test pattern
      clr.l   d2                ;Address pattern
      move.w  #7,d0             ;Context
@11   bsr     SetContext
      move.w  #127,d1           ;Segment
@12   bsr     SetSegment
      move.b  d2,(a1)           ;Write to Limit register
```

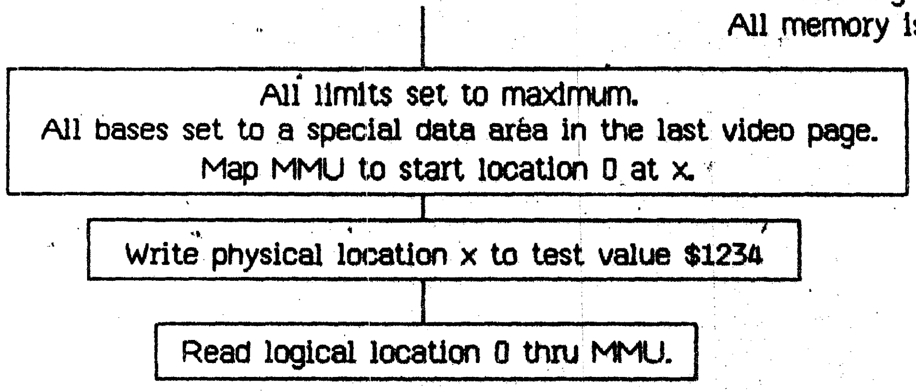
```
        add.w   #$0101,d2           ;Next address
        move.w  d2,(a0)             ;Write to Base register
        add.w   #$0101,d2           ;Next address
        sub.w   #1,d1               ;Next segment
        bpl     @12
        sub.w   #1,d0               ;Next context
        bpl     @11
;
;
;        Loop thru all contexts and segments, read and compare data.
        clr.l   d2                  ;Address pattern
        move.w  #7,d0               ;Context
@13     bsr     SetContext
        move.w  #127,d1             ;Segment
@14     bsr     SetSegment
        move.b  (a1),d3             ;Read Limit register
        cmp.b   d2,d3               ;...same as written?
        beq     @15                 ;...yes, continue
        bsr     BADLIMIT            ;...no, flag failure
;
@15     add.w   #$0101,d2           ;Next address
        move.w  (a0),d3             ;Read Base register
        cmp.w   d2,d3               ;...same as written?
        beq     @16                 ;...yes, continue
        bsr     BADBASE             ;...no, flag failure
;
@16     add.w   #$0101,d2           ;Next address
        sub.w   #1,d1               ;Next segment
        bpl     @14
        sub.w   #1,d0               ;Next context
        bpl     @13
        rts
;
;_______________________________________________________________
;
; Moving Inversions test.
;
;
; Example:
;
; Low to high pass
; 1. Write background pattern
; 2. For every (1) addr, low to high, do Read/Verify, Write complememt, Read/Verify
; 3. Two (2) passes, for every other address (first even, second odd)
     do Read/Verify, Write complememt, Read/Verify
; 4. Four (4) passes, for every 4th address (first even, second odd,...)
;    do Read/Verify, Write complememt, Read/Verify
; 5. Eight (8) passes, for every 8th address (first even, second odd,...)
     do Read/Verify, Write complememt, Read/Verify
; 6.  .... until step count equals one half the size of the memory.
;
; High to low pass
; 1. Write background pattern
; 2. For every (1) addr, high to low, do Read/Verify, Write complememt, Read/Verify
; 3. Two (2) passes, for every other address (first even, second odd)
     do Read/Verify, Write complememt, Read/Verify
; 4. Four (4) passes, for every 4th address (first even, second odd,...)
;    do Read/Verify, Write complememt, Read/Verify
; 5. Eight (8) passes, for every 8th address (first even, second odd,...)
     do Read/Verify, Write complememt, Read/Verify
; 6.  .... until step count equals one half the size of the memory.
;
;
; Step     (1)          (2)           (4)
; Pass      1            1  2          1  2  3  4
; Loc   0  xx           xx            xx
;       1  xx              xx            xx
;       2  xx           xx                  xx
;       3  xx              xx                  xx
;       4  xx           xx            xx
;       5  xx              xx            xx
;       6  xx           xx                  xx
;       7  xx              xx                  xx
;       8  xx           xx            xx
;       9  xx              xx            xx
;      10  xx           xx                  xx
;      11  xx              xx                  xx
;
;
;  Total passes = 10 for 1K RAM,
;   Increments: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512
;
;   Increment = x
;   Passes    = x
;   Expect    = $F0F0    All ones and all zeros for each 1Kx4 RAM
;   Compl Exp = $0F0F
;
;
;
;   Write all of MMU to $F0F0
;
;   Total Pass Loop
;
;   Pass loop
;         Read data.
;         Compare to expect and log failures.
;          Write compl expect data.
;          Read data.
;          Compare to expect and log failures.
;
;          Add increment to address under test
;          Check if out of bounds
;          ...No, go to top of Pass loop
;          ...Yes, Continue
;
;          Decrement pass counter
;
;   End of Pass loop
;          Passes all done?
;          ...No, Start address at next top address and go to top of Pass loop
;          ...Yes, Continue
;
;   Decrement Total Pass counter
;   End of all passes?
;   ...No, complement both Expect Data, Go to top of Total Pass Loop
;   ...Yes, continue.
;
;   Zero all of MMU
;
;
;==========================================================================;
;
INVERSIONS
     move.l   #$F0F0F0F0,d2    ;Write all of MMU to $F0F0
     bsr      WPAT
;
     move.l   #$0F0FF0F0,d2    ;Both patterns
     move.w   #10,d6           ;Total Pass Loop
     lea      INCRE,a3         ;Increment table
;
@1   move.w   (a3),d5          ;Increment and pass counter
;   Pass loop
     clr.l    d7               ;Current Address
;
@2   bsr      ITEST
     add.w    d0,d7            ;Add increment to address under test
     cmp.w    #1000,d7         ;Check if out of bounds
     bne      @2               ;...No, go to top of Pass loop
;                               ...Yes, Continue
     sub.w    #1,d5            ;Decrement pass counter
;  End of Pass loop
     cmp.w    #0,d5            ;Passes all done?
     beq      @3               ;...Yes, Continue
     clr.l    d7               ;Start address at top again
     move.w   (a3),d0          ;... increment amount
     add.w    d0,d7
     sub.w    d5,d7
     bra      @2
;
@3   sub.w    #1,d6            ;Decrement Total Pass counter
     cmp.w    #0,d6            ;End of all passes?
     beq      IDECREMENT       ;...Yes, continue.
;
     eor.l    #$ffffffff,d2    ;...No, complement both Expect Data,
     adda     #2,a3            ;...next increment value
     jmp      @1               ;Go to top of Total Pass Loop
;
;
;
; now we start it all over, only decrementing the address this time . . .
;
IDECREMENT
     move.l   #$F0F0F0F0,d2    ;Write all of MMU to $F0F0
     bsr      WPAT
;
     move.l   #$0F0FF0F0,d2    ;Both patterns
     move.w   #10,d6           ;Total Pass Loop
     lea      INCRE,a3         ;Increment table
;
@1   move.w   (a3),d5          ;Decrement and pass counter
;   Pass loop
     move.l    #999,d7         ;Current Address
;
@2   bsr      ITEST
     sub.w    d0,d7            ;Add increment to address under test
     bne      @2               ;...No, go to top of Pass loop
;                               ...Yes, Continue
     sub.w    #1,d5            ;Decrement pass counter
;  End of Pass loop
     cmp.w    #0,d5            ;Passes all done?
     beq      @3               ;...Yes, Continue
     move.l   #999,d7          ;Start address at top again
     move.w   (a3),d0          ;... increment amount
     sub.w    d0,d7
     add.w    d5,d7
     bra      @2
;
@3   sub.w    #1,d6            ;Decrement Total Pass counter
     cmp.w    #0,d6            ;End of all passes?
     beq      @4               ;...Yes, continue.
;
     eor.l    #$ffffffff,d2    ;...No, complement both Expect Data,
        adda    #2,a3              ;...next increment value
        jmp     @1                 ;Go to top of Total Pass Loop
;
@4      clr.l   d2                 ;Zero all of MMU
        bsr     WPAT
        rts
;
;
;
ITEST
@2      bsr     GetAddress         ;Get address based on d7, in a0 and a1
;
        move.w  (a0),d0            ;Read Base data.
        cmp.w   d0,d2              ;Compare to expect and log failures.
        beq     @3
        bsr     BADBASE
;
@3      swap    d2                 ;Now complement data
        move.w  d2,(a0)            ;Write complement expect data.
        move.w  (a0),d0            ;Read Base data.
        cmp.w   d0,d2              ;Compare to expect and log failures.
        beq     @4
        bsr     BADBASE
;
@4      swap    d2                 ;Now original data
        move.b  (a1),d0            ;Read Limit data.
        cmp.b   d0,d2              ;Compare to expect and log failures.
        beq     @5
        bsr     BADLIMIT
;
@5      swap    d2                 ;Now complement data
        move.b  d2,(a1)            ;Write complement expect data.
        move.b  (a1),d0            ;Read Limit data.
        cmp.b   d0,d2              ;Compare to expect and log failures.
        beq     @6
        bsr     BADLIMIT
;
@6      swap    d2                 ;Now original data
        move.w  (a3),d0
        rts
;
INCRE .WORD  1, 2, 4, 8, 16, 32, 64, 128, 256, 512
;___________________________________________________________________________
;
; if location <> written data then flag as bad RAM
; D2 has written data, D3 has read data
;
; Do whole word for base register.
BADBASE
        move.w  d3,d6
        eor.w   d2,d6              ;Find failing bits
        move.w  d6,d7
        and.w   #$000F,d6
        beq     @1
        lea     BASE1,a6
        move.w  #$0123,(a6)
;
@1      move.w  d7,d6
        and.w   #$00F0,d6
        beq     @2
        lea     BASE2,a6
        move.w  #$4567,(a6)
;
@2      move.w  d7,d6
        and.w   #$0F00,d6
        beq     @3
        lea     BASE3,a6
        move.w  #$89AB,(a6)
;
@3      move.w  d7,d6
        and.w   #$F000,d6
        beq     @4
        lea     BASE4,a6
        move.w  #$CDEF,(a6)
;
@4      rts
;
; Do lower byte only for limit register
BADLIMIT
        move.b  d3,d6
        eor.b   d2,d6
        move.b  d6,d7
        and.b   #$0F,d6            ;Lower OK?
        beq     @1
        lea     LIMIT1,a6
        move.w  #$0123,(a6)
;
@1      move.b  d7,d6
        and.b   #$F0,d6
        beq     @2
        lea     LIMIT2,a6
        move.w  #$4567,(a6)
;
@2      rts
;___________________________________________________________________________
;
```

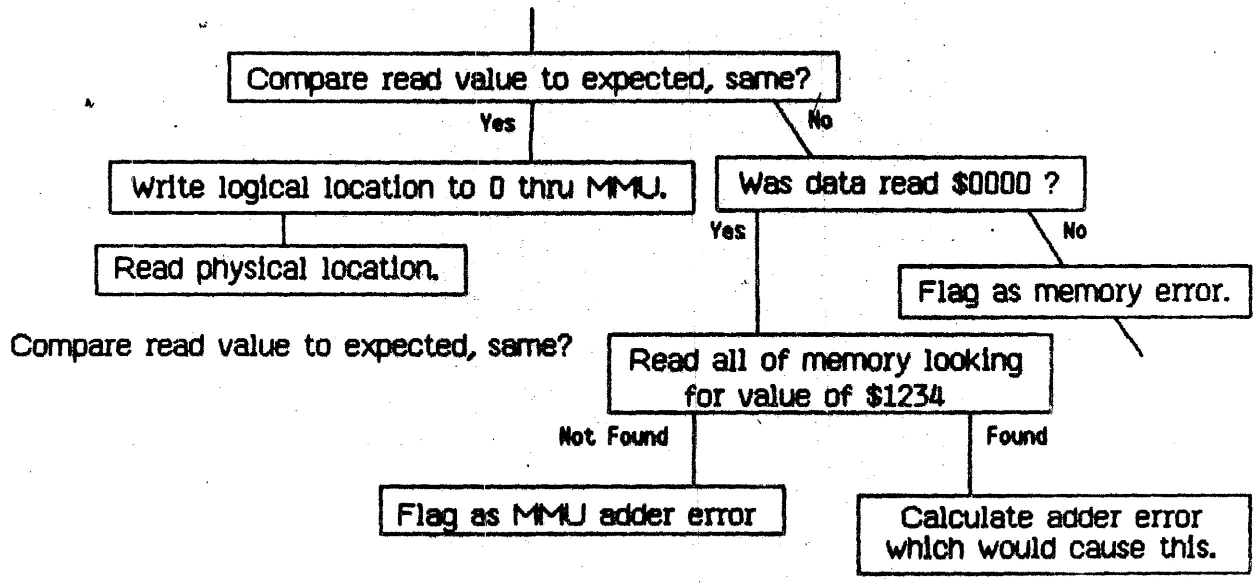
```
; Sets to requested context, number 0 to 7 in d0
;
SETCONTEXT
     and.w   #$7,d0          ;Make it legal
;
     tst.w   $0003800        ;SEG0 to a 0
     tst.w   $0003A00        ;SEG1 to a 0
     tst.w   $0003C00        ;SEG2 to a 0
;
     cmp.w   #0,d0
     bne     @1
     bra     @10
;
@1   cmp.w   #1,d0
     bne     @2
     tst.w   $0003900        ;SEG0 to a 1
     bra     @10
;
@2   cmp.w   #2,d0
     bne     @3
     tst.w   $0003B00        ;SEG1 to a 1
     bra     @10
;
@3   cmp.w   #3,d0
     bne     @4
     tst.w   $0003900        ;SEG0 to a 1
     tst.w   $0003B00        ;SEG1 to a 1
     bra     @10
;
@4   tst.w   $0003D00        ;SEG2 to a 1
     cmp.w   #4,d0
     bne     @5
     bra     @10
;
@5   cmp.w   #5,d0
     bne     @6
     tst.w   $0003900        ;SEG0 to a 1
     bra     @10
;
@6   cmp.w   #6,d0
     bne     @6
     tst.w   $0003B00        ;SEG1 to a 1
     bra     @10
;
@7   tst.w   $0003900        ;SEG0 to a 1
     tst.w   $0003B00        ;SEG1 to a 1
;
@10  rts
;_______________________________________________________________________
;
; Sets segment address for Limit register in a3, segment requested in d1.
; Uses d5 as scratch
;
SetSegment
     move.l  #$010000,a1     ;Address of first Limit register
     move.w  d1,d5
     and.l   #$7F,d5         ;Make legal
     swap    d5              ;Now in bits 15 to 21
     asl.l   #2,d5           ;Now in bits 17 to 23
     adda    d5,a1           ;Final address
     move.l  a1,d5
     or.l    #$008000,d5
     move.l  d5,a0
     rts
;_______________________________________________________________________
;
; Get address based on d7
; Returns addresses in a0 and a1
GetAddress
     clr.l   d6              ;Current counter
     clr.w   d0              ;Context
@1   bsr     SetContext
     clr.w   d1              ;Segment
@2   bsr     SetSegment
     cmp.w   d6,d7           ;At requested?
     beq     @3
     add.w   #1,d6
     add.w   #1,d1           ;Next segment
     cmp.w   #128,d1         ;..done all?
     bne     @2
     add.w   #1,d0           ;Next context
     cmp.w   #8,d0           ;..done all?
     bne     @1
@3   rts
;_______________________________________________________________________
;
; Table of bad RAMs, 0 means good, other means failure
; (Table must be grouped for clear to work properly)
BASE4   .WORD    0      ;Base register bits 12-15
BASE3   .WORD    0      ;Base register bits  8-11
BASE2   .WORD    0      ;Base register bits  4- 7
BASE1   .WORD    0      ;Base register bits  0- 3
LIMIT2  .WORD    0      ;Base register bits  4- 7
LIMIT1  .WORD    0      ;Base register bits  0- 3
;_______________________________________________________________________
;
```

```
;
BMMUSAVE        ; The data following is the current state of the MMU before the test
;
BEND            ; This is the end of the test that is copied to video memory
;
BSAVE           ; This is the start of the saved video memory area.
;
         .END
```

MMU Functional

Assuming physical starts at x.
All memory is assumed written to 0.


All limits set to maximum.
All bases set to a special data area in the last video page.
Map MMU to start location 0 at x.
Write physical location x to test value $1234
Read logical location 0 thru MMU.


Compare read value to expected, same?
Yes
No
Write logical location to 0 thru MMU.
Read physical location.
Compare read value to expected, same?
Was data read $0000 ?
Yes
No
Flag as memory error.
Read all of memory looking for value of $1234
Not Found
Found
Flag as MMU adder error
Calculate adder error which would cause this.

Data area in last video page is set to a specific pattern.

The data area pattern is read thru all segment addresses.

Base address adders.

	Base	Physical
Sum 4, COut	$00080000	$00000000
	$00070001	$0000FFFF
	$000F0001	$0000FFFF
Sum 3	$00040000	$00000000
	$00030001	$0000FFFF
	$00070001	$0000FFFF
Sum 2	$00020000	$00000000
	$00010001	$0000FFFF
	$00030001	$0000FFFF
Sum 1	$00010000	$00000000
	$00000001	$0000FFFF
	$00010001	$0000FFFF

	Base	Physical
Sum 4, COut	$00008000	$00000000
	$00000000	$00008000
	$00008000	$00008000
Sum 3	$00004000	$00000000
	$00000000	$00004000
	$00004000	$00004000
Sum 2	$00002000	$00000000
	$00000000	$00002000
	$00002000	$00002000
Sum 1	$00001000	$00000000
	$00000000	$00001000
	$00001000	$00001000

	Base	Physical
Sum 4, COut	$00000800	$00000000
	$00000000	$00000800
	$00000800	$00000800
Sum 3	$00000400	$00000000
	$00000000	$00000400
	$00000400	$00000400
Sum 2	$00000200	$00000000
	$00000000	$00000200
	$00000200	$00000200
Sum 1	$00000000	$00000000
	$00000100	$00000000
	$00000000	$00000100
	$00000100	$00000100

```
;  Edit Date:  04/12/83
;
;
;==========================================================================
;
;  FILE NAMES:
;       TWOP.HDR.TEXT
;       TWOP.ASM.TEXT
;       TWOPORT (code file)
;
;  FUNCTION:
;       Verify as much of the TWO Port Card as possible.
;
;  LOGIC TESTED:
;       Portions of the Two Port Card, as described below.
;
;  TECH OPTIONS:
;   The following options are supported (1-3 are standard):
;       0:  normal program execution
;       1:  loop until NMI (error count kept in ERRCNT, loop count in OUTCNT)
;       2:  loop INCNT number of times (or until error)(loop count in OUTCNT)
;       3:  loop on test until error (loop count kept in OUTCNT),
;             exit test at error without resetting cause of error
;
;      10: Loop on sending data thru PA0-PA7 from port A to port B
;      11: Loop on sending data thru PA0-PA7 from port B to port A
;      12: Loop on sending from PB7(CRES) to PB0(OCD) from port A to port B
;      13: Loop on sending from PB7(CRES) to PB0(OCD) from port B to port A
;      14: Loop on sending from PB4(CMD) to PB6(PARITY) from port A to port B
;      15: Loop on sending from PB4(CMD) to PB6(PARITY) from port B to port A
;      16: Loop on sending from PB3(R/W) to CB1(CHK) from port A to port B
;      17: Loop on sending from PB3(R/W) to CB1(CHK) from port B to port A
;      18: Loop on sending from CA2(STRB) to PB1/CA1(BSY) from port A to port B
;      19: Loop on sending from CA2(STRB) to PB1/CA1(BSY) from port B to port A
;      20: Loop on sending from port B to test Port A parity circuit.
;      21: Loop on sending from port A to test Port B parity circuit.
;
;      32:  Loop at end of test, for program debug use.
;      33:  Ignore any timer test failures, gather data for debug use.
;
;  ERROR CODES:
;
;++ Flag for document extractor
;
;  Step:
;       1 - Initial test setup, hardware and software.
;       2 - Check data lines to Two Port Card.
;       3 - Check interrupt capability of Two Port Card.
;       4 - Port A 6522(VIA) register test.
;       5 - Port B 6522(VIA) register test.
;       6 - Port A 6522(VIA), Timer #1 test.
;       7 - Port A 6522(VIA), Timer #2 test.
;       8 - Port B 6522(VIA), Timer #1 test.
;       9 - Port B 6522(VIA), Timer #2 test.
;      10 - PA0-PA7 data lines thru loopback connector test, both ports.
;      11 - PB0,PB7 data lines thru loopback connector test, both ports.
;      12 - PB4,PB6 data lines thru loopback connector test, both ports.
;      13 - PB3,CB1 data lines thru loopback connector test, both ports.
;      14 - CA2,PB1 data lines thru loopback connector test, both ports.
;      15 - Port A Parity circuit test.
;      16 - Port B Parity circuit test.
;
;
;  Errors:
;       0 - no error
;       1 - Could not locate Two Port Card in any expansion slot.
;       2 - Data read is not the same as data written to register on VIA.
;       3 - Card could not supply an interrupt (possibly timer #1 also).
;       4 - 6522(VIA) Timer failure. Timer too fast, time is too short.
;       5 - 6522(VIA) Timer failure. Timer too slow, time is too fast.
;       6 - Data line PA0-PA7 failed wraparound test, port A driving port B,
;           or no Loop-Back connector installed.
;       7 - Data line PA0-PA7 failed wraparound test, port B driving port A.
;       8 - OCD(PB0) or RESET/(PB7) failed, port A driving port B.
;       9 - OCD(PB0) or RESET/(PB7) failed, port B driving port A.
;      10 - CMD(PB4) or PARITY(PB6) failed, port A driving port B.
;      11 - CMD(PB4) or PARITY(PB6) failed, port B driving port A.
;      12 - Interrupt occurred when none expected, on Port A.
;      13 - Interrupt occurred when none expected, on Port B.
;      14 - Expected interrupt did not occur on port B.
;      15 - Expected interrupt did not occur on port A.
;      16 - STRB(CA2) or BSY(PB1) failed, port A driving port B.
;      17 - STRB(CA2) or BSY(PB1) failed, port B driving port A.
;      18 - PB1 on port B failure (BSY), port A driving port B.
;      19 - PB1 on port A failure (BSY), port A driving port B.
;      20 - Bad data (PA0-PA7) during parity test of Port A.
;      21 - Parity circuit for port A gives wrong summation result (CB2).
;      22 - Bad data (PA0-PA7) during parity test of Port A.
;      23 - Parity circuit for port A gives wrong summation result (CB2).
;
;
;++ Flag for document extractor
;
;   HARDWARE:   Working LISA, I/O Port Test Card (cables connected or not).
;
;
```

```
;   THEORY OF TEST OPERATION:
;
;
;       Register Read/Write test of the VIA (6522).  This test verifies that most
;               if the registers in the 6522 are read/write-able.  This test
;               does not test all registers since some registers are read
;               or write only.
;               Details:
;                  a) All registers are initialized as follows.
;
;
;       Timer test for the 6522.  Timer #1 and Timer #2 are tested to assure that
;               interrupts are generated at the end of the programmed time.
;
;
;
;   WRITTEN BY:  REV 1.0 G.Cossey  01/21/83
;
;   MODIFIED:    REV 1.1 G.Cossey  04/12/83, now uses library routines. Removed
;                code to run on APPLE II, now only from LISA.
;
;   TO BE DONE:
;
;============================================================================;
        .ABSOLUTE
        .PROC   TWOPORT,0
;
;==============================;
; TRAP AND ERROR VECTOR LOCATIONS
;==============================;
BUSVCT  .EQU    $0008           ; bus error
ADRVCT  .EQU    $000C           ; address error
ILLVCT  .EQU    $0010           ; illegal instruction
AXXVCT  .EQU    $0028           ; future traps (emulator)
FXXVCT  .EQU    $002C
;==============================;
; INTERNAL INTERRUPT VECTORS    ;
;==============================;
LEV1VCT .EQU    $0064           ; floppy, parallel port, VTIR
LEV2VCT .EQU    $0068           ; keyboard interface VIA
LEV3VCT .EQU    $006C           ; slot 2
LEV4VCT .EQU    $0070           ; slot 1
LEV5VCT .EQU    $0074           ; slot 0
LEV6VCT .EQU    $0078           ; RS232 chip
NMIVCT  .EQU    $007C           ; NMI
;
;==============================;
; now, the local equates . . .  ;
;==============================;
;
ORB             equ     $00     ;       port B
ORA             equ     $08     ;       port A
DDRB            equ     $10     ;Data direction of port B
DDRA            equ     $18     ;                  port A
T1CL            equ     $20     ;T1 Counter (low)
T1CH            equ     $28     ;           (high)
T1LL            equ     $30     ;T1 Latch (low)
T1LH            equ     $38     ;         (high)
T2CL            equ     $40     ;T2 Counter
T2CH            equ     $48
SHR             equ     $50     ;Shift register
ACR             equ     $58     ;Aux control
PCR             equ     $60     ;Peripheral control
IFR             equ     $68     ;Int. flags
IER             equ     $70     ;interrupt enable
;
;
;For Keyboard COPS use.
KBDVIA          equ     $fcdd80 ;Base address for keyboard VIA 6522
KIER            equ     29      ;interrupt enable
;
;==============================;
; START - PROGRAM ENTRY POINT   ;
;==============================;
        .ORG    $0800           ; we are normally loaded at $800 (we try to be
                                ;   relocatable, though)
START   BRA.S   START1          ; skip around parameter area
;
REVDATE .WORD   11,0            ;Revision number (use decimal numbers for Pascal)
;                                ... x.x > xx, so REV 1.0 is 10 here.
                                ; label "TECH" is location START +6 ($806)
                                ; note - parameters are word-length (2 bytes)
                                ;
TECH    .WORD   0               ; non-zero values select tech-mode options
                                ;
                                ; the next three parameter words are used only
                                ;  if the tech mode is non-zero.
                                ;
ERRCNT  .WORD   0               ; exit parm - holds number of errors encountered
OUTCNT  .WORD   0               ; exit parm - holds number of passes completed
INCNT   .WORD   0               ; entry parm - holds number of times to execute prog
RESVD   .WORD   0               ; reserved for later expansion
;
IERRCNT .EQU    2
IOUTCNT .EQU    4
IINCNT  .EQU    6
```

```
;
;==============================;
; INPUT PARAMETERS - START + $010
;==============================;
INBUF0  .WORD   0               ; currently unused
INBUF1  .WORD   0
INBUF2  .WORD   0
INBUF3  .WORD   0
INBUF4  .WORD   0
INBUF5  .WORD   0
INBUF6  .WORD   0
INBUF7  .WORD   0
;==============================;
; RESULT BUFFER - START + $020 ;
;==============================;
DREGS   .WORD   0,0             ; (D0) :
        .WORD   0,0             ; (D1) :
        .WORD   0,0             ; (D2) :
        .WORD   0,0             ; (D3) :
        .WORD   0,0             ; (D4) :
        .WORD   0,0             ; (D5) :
        .WORD   0,0             ; (D6) : step number
        .WORD   0,0             ; (D7) : error code
;
; AREGS - START + $040
AREGS   .WORD   0,0             ; (A0) :
        .WORD   0,0             ; (A1) :
        .WORD   0,0             ; (A2)
        .WORD   0,0             ; (A3)
        .WORD   0,0             ; (A4) :
        .WORD   0,0             ; (A5)
        .WORD   0,0             ; (A6) : obliterated restoring environment
        .WORD   0,0             ; (A7) : stack pointer
;
; EXCEPTION INFO - START + $060
; GROUP 0
EXCFC   .WORD   0               ; function code
EXCADR  .WORD   0,0             ; address
EXCIR   .WORD   0               ; instruction register
; GROUP(s) 0 & 1
EXCSR   .WORD   0               ; status register
EXCPC   .WORD   0,0             ; program counter
OLDPC   .WORD   0,0
OLDSP   .WORD   0,0
OLDSR   .WORD   0,0
;
;==============================;
; NOW CONTINUE WITH THE CODE   ;
;==============================;
START1  LEA     OLDPC,A0        ;
        MOVE.L  (SP)+,(A0)      ; save return address, set SP to top of
        LEA     OLDSP,A0        ;  new stack
        MOVE.L  SP,(A0)         ; then save it also
        LEA     START,SP        ; place our stack right below the program
;
        LEA     OLDSR,A0        ;Save status reg value
        MOVE.L  SR,(A0)
        ORI.W   #$0700,SR       ;Disable interrupts
;
        LEA     MISC,A0         ; initialize interrupt and trap vectors in case
        MOVEA   #0,A1           ;  of unexpected exceptions
        LEA     OLDVCTR,A2      ;also save incoming vectors
        MOVEQ   #64,D0
@1
        MOVE.L  (A1),(A2)+      ;save incoming vectors in array OLDVCTR
        MOVE.L  A0,(A1)+        ; first set all vectors to point to miscelaneous
        SUBQ    #1,D0           ;  exception handler . . .
        BGT     @1
;
        LEA     BUSERR,A0       ; then set up special vectors for the more
        MOVE.L  A0,BUSVCT       ;  common exceptions . . .
        LEA     ADRERR,A0
        MOVE.L  A0,ADRVCT
        LEA     ILLERR,A0
        MOVE.L  A0,ILLVCT
        LEA     LEV1INT,A0
        MOVE.L  A0,LEV1VCT
        LEA     LEV2INT,A0
        MOVE.L  A0,LEV2VCT
        LEA     LEV3INT,A0
        MOVE.L  A0,LEV3VCT
        LEA     LEV4INT,A0
        MOVE.L  A0,LEV4VCT
        LEA     LEV5INT,A0
        MOVE.L  A0,LEV5VCT
        LEA     LEV6INT,A0
        MOVE.L  A0,LEV6VCT
        LEA     NMI,A0
        MOVE.L  A0,NMIVCT
;
;
        CLR.L   D0              ; clear all the registers except A7
        CLR.L   D1
        CLR.L   D2
        CLR.L   D3
        CLR.L   D4
        CLR.L    D5
        CLR.L    D6
        CLR.L    D7
        MOVEA.L  D0,A0
        MOVEA.L  D0,A1
        MOVEA.L  D0,A2
        MOVEA.L  D0,A3
        MOVEA.L  D0,A4
        MOVEA.L  D0,A5
        MOVEA.L  D0,A6
        BRA      MAIN               ; and branch to the main routine
;
;==============================;
; DEFAULT EXCEPTION HANDLERS    ; these exception handlers just set an error
;==============================;  code and exit . . .
MISC    MOVEQ    #$50,D7            ; miscellaneous exceptions (traps, divide by 0,
        BRA.S    EXCP               ;   etc. )
BUSERR  MOVEQ    #$51,D7            ; bus error exception (bus timeout, invalid MMU
        BRA.S    EXCP0              ;   access)
ADRERR  MOVEQ    #$52,D7            ; address error
        BRA.S    EXCP0
ILLERR  MOVEQ    #$53,D7            ; illegal instruction error
        BRA.S    EXCP
;
LEV1INT MOVEQ    #$71,D7            ; level 1 interrupt (parallel port, vertical
        BRA.S    EXCP               ;   retrace)
LEV2INT MOVEQ    #$72,D7            ; level 2 interrupt (keyboard)
        BRA.S    EXCP
LEV3INT MOVEQ    #$73,D7            ; level 3 interrupt (expansion slot 2)
        BRA.S    EXCP
LEV4INT MOVEQ    #$74,D7            ; level 4 interrupt (expansion slot 1)
        BRA.S    EXCP
LEV5INT MOVEQ    #$75,D7            ; level 5 interrupt (expansion slot 0)
        BRA.S    EXCP
LEV6INT MOVEQ    #$76,D7            ; level 6 interrupt (RS-232 ports)
        BRA.S    EXCP
NMI     MOVEQ    #$77,D7            ; level 7 (NMI) interrupt (parity error)
        BRA.S    EXCP
;
EXCP0   LEA      EXCFC,A6           ; NOTE A6 IS DESTROYED HERE
        MOVE     (SP)+,(A6)+        ; save extra info for group 0
        MOVE.L   (SP)+,(A6)+
        MOVE     (SP)+,(A6)+
EXCP    LEA      EXCSR,A6
        MOVE     (SP)+,(A6)+        ; save common exception info
        MOVE.L   (SP)+,(A6)+
;
;       EXIT. Save current register contents in save area and return to caller
;
EXIT
        ORI.W    #$0700,SR          ; disable interrupts
        LEA      DREGS,A6
        MOVEM.L  D0-D7/A0-A7,(A6) ; push return registers in result buffer
;
        LEA      OLDVCTR,A0         ;restore incoming exception vectors
        MOVEA    #0,A1
        MOVEQ    #64,D0
@1      MOVE.L   (A0)+,(A1)+
        SUBQ     #1,D0
        BGT.S    @1
;
        LEA      OLDSR,A0           ;Restore incoming status register
        MOVE.L   (A0),SR
;
        LEA      OLDSP,A6           ; get back incomming stack pointer
        MOVE.L   (A6),SP
        LEA      OLDPC,A6           ; and program counter
        MOVE.L   (A6),-(SP)
        RTS
;
OLDVCTR .BLOCK   256,0              ;storage area for incoming exception vectors
;
        .INCLUDE  lib:COPSCMD.TEXT
        .INCLUDE  lib:FINDC.ASM.TEXT
        .INCLUDE  TWOP.ASM.TEXT
;
        .END
;
```

```
;  Edit Date:  04/12/83
;==========================================================================
;
; File:   TWOP.ASM.TEXT
;
; NOTE: Refer to TWOP.HDR.TEXT for program details.
;
;==========================================================================
;
TICKTOCK        equ     0       ;Timer counter
T1              equ     2       ;Timer 1 happened flag
T2              equ     4       ;Timer 2 happened flag
KBDIEN          equ     6       ;Keyboard VIA interrupt enable value
SAVEVECTOR      equ     8       ;Interrupt vector address, original
WhichTimer      equ     12      ;Flag for timer under test
CA1VIA1         equ     14
CA1VIA2         equ     16
CB1VIA1         equ     18
CB1VIA2         equ     20
ParityTests     equ     22
;
; ___________________________________________________________________
; ___________________________________________________________________
;
;    a0 - Scratch                       d0 - Scratch
;    a1 - Scratch                       d1 - Scratch
;    a2 - First VIA address.            d2 - Scratch
;    a3 - Second VIA address.           d3 - Scratch
;    a4 - Interrupt vector address      d4 - Scratch
;    a5 - LOCAL variable table.         d5 - Tech Mode
;    a6 - DEBUG                         d6 - Step number
;    a7 - Stack                         d7 - Error code
; ___________________________________________________________________
; ___________________________________________________________________
;
;
MAIN    clr.w   d7              ;Clear error flag
        move.w  #1,d6           ;Init to step #1
        lea     LOCAL,a5        ;LOCAL table pointer
        lea     DEBUG,a6
;
        move.b  #$00,d1         ;Boot ID looking for
        move.b  #$02,d2
        bsr     FINDC
        cmp.w   #0,d0           ;...find it?
        beq     @105            ;...no, exit with error

        lea     INTCARD,a1
        move.l  2(a3),(a1)      ;...address of interrupt vector
        move.l  2(a3),a4
        lea     CARD,a1         ;Base address
        move.l  6(a3),(a1)      ;...address of VIA
;
        move.l  (a1),a2         ;First VIA
        adda    #$2001,a2
        move.l  a2,(a6)         ;for debug
        move.l  (a1),a3         ;Second VIA
        adda    #$2801,a3
        move.l  a3,6(a6)        ;for debug
;
        lea     TECH,a1
        move.w  (a1),d5         ;Get Tech Mode
;
        bsr     DISABLEOTHER    ;Disable interrupts from other sources
;
@1      clr.w   d7              ;Clear fail flag.
;
        cmp.w   #10,d5          ;See if special tech mode loops wanted
        beq     @2
        cmp.w   #11,d5
        beq     @2
        cmp.w   #12,d5
        beq     @3
        cmp.w   #13,d5
        beq     @3
        cmp.w   #14,d5
        beq     @4
        cmp.w   #15,d5
        beq     @4
        cmp.w   #16,d5
        beq     @5
        cmp.w   #17,d5
        beq     @5
        cmp.w   #18,d5
        beq     @6
        cmp.w   #19,d5
        beq     @6
        cmp.w   #20,d5
        beq     @7
        cmp.w   #21,d5
        beq     @8
;
        move.w  #2,d6           ;Step #2
        bsr     CHECKLINE       ;Check data line connection to card.
        cmpi.w  #0,d7           ;...any errors?
        bne     @10             ;...exit on error
```

[illegible]

```
;
        move.w  #3,d6           ;Step #3
        bsr     CKINTERRUPT     ;Check can get interrupt from TWO Port card.
        cmpi.w  #0,d7           ;...any errors?
        bne     @10             ;...exit on error
;
        move.w  #4,d6           ;Step #4
        bsr     CK1VIA           ;Check registers on first 6522(VIA)
        cmpi.w  #0,d7           ;...any errors?
        bne     @10             ;...exit on error
;
        move.w  #5,d6           ;Step #5
        bsr     CK2VIA          ;Check registers on second VIA
        cmpi.w  #0,d7           ;...any errors?
        bne     @10             ;...exit on error
;
        move.w  #6,d6           ;Step #6,7
        bsr     CK1TIMER         ;Check timers on 6522(VIA)
        cmpi.w  #0,d7           ;...any errors?
        bne     @10             ;...exit on error
;
        move.w  #8,d6           ;Step #8,9
        bsr     CK2TIMER         ;Check timers on 6522(VIA)
        cmpi.w  #0,d7           ;...any errors?
        bne     @10
;
@2      move.w  #10,d6          ;Step #10
        bsr     PA0to7          ;Check connection thru cable, PA0-PA7
        cmpi.w  #0,d7           ;...any errors?
        bne     @10
;
@3      move.w  #11,d6          ;Step #11
        bsr     PB0and7         ;Check connection thru cable PB0,PB7
        cmpi.w  #0,d7           ;...any errors?
        bne     @10
;
@4      move.w  #12,d6          ;Step #12
        bsr     PB4and6         ;Check connection thru cable PB4,PB6
        cmpi.w  #0,d7           ;...any errors?
        bne     @10
;
@5      move.w  #13,d6          ;Step #13
        bsr     PB3andCB1       ;Check connection thru cable PB3,CB1
        cmpi.w  #0,d7           ;...any errors?
        bne     @10
;
@6      move.w  #14,d6          ;Step #14
        bsr     CA2andPB1       ;Check connection thru cable CA2,PB1
        cmpi.w  #0,d7           ;...any errors?
        bne     @10
;
@7      move.w  #15,d6          ;Step #15
        bsr     AParity         ;Check Port A parity circuit
        cmpi.w  #0,d7           ;...any errors?
        bne     @10
;
@8      move.w  #16,d6          ;Step #16
        bsr     BParity         ;Check Port B parity circuit
        cmpi.w  #0,d7           ;...any errors?
        bne     @10
;
;
@10     bsr     RESTORE         ;Restore any disabled functions
;
        lea     TECH,a1
        cmpi.w  #0,d7           ;test for errors
        bne     @100            ;...toggle pass counter?
        addq.w  #1,IOUTCNT(a1)  ;...yes, Increment pass counter.
        bra     @101
@100    addq.w  #1,IERRCNT(a1)
@101    subq.w  #1,IINCNT(a1)   ;Decrement test limit counter
;
        cmp.w   #1,d5           ;See if Tech mode 1
        beq     MAIN            ;...yes, loop forever
;
        cmp.w   #2,d5           ;See if Tech mode 2
        bne     @102            ;...no
        cmp.w   #0,d7           ;...Error occured?
        bne     @103            ;...yes, abort test
        cmp.w   #0,IINCNT(a1)   ;...test count expired?
        bne     MAIN            ;...no, continue.
;
@102    cmp.w   #3,d5           ;See if Tech mode 3
        bne     @103            ;...no
        cmp.w   #0,d7           ;...Error occured?
        bne     @103            ;...yes, abort test
        bra     MAIN
;
@103    cmpi.w  #$20,d5         ;Tech Mode, loop at end of test?
        beq     @103
; Normal exit
@104    bra     EXIT
;
; Error exit, could not find card
@105    move.w  #1,d7
        bra     @104
```

```
;
;=============================================================================
;=============================================================================
;
; Disable interrupts from other sources
;
DISABLEOTHER  ori #$0700,sr     ; mask all interrupts
        lea     KBDVIA,a1       ;Get Keyboard VIA address
        move.b  KIER(a1),d1     ;...Get interrupt enable for save
        ori.b   #$80,d1         ;...add in set bit
        move.b  d1,KBDIEN(a5)   ;Place in LOCAL table
        move.b  #$7f,KIER(a1)   ;Disable interrupts
        rts
;
;=============================================================================
;
; Init I/O Port Test Card to known state.
;
INITBOARD clr.b d1              ;Init to zero
        move.b  #$7f,IER(a2)    ;All interrupts off
        move.b  d1,IFR(a2)
        move.b  d1,ACR(a2)      ;Aux control
        move.b  d1,PCR(a2)      ;Per control
        move.b  d1,ORB(a2)      ;Ports
        move.b  d1,ORA(a2)
        move.b  d1,DDRB(a2)     ;Direction
        move.b  d1,DDRA(a2)
        move.b  d1,T1CL(a2)     ;Timer #1
        move.b  d1,T1CH(a2)
        move.b  d1,T1LL(a2)     ;Latch
        move.b  d1,T1LH(a2)
        move.b  d1,T2CL(a2)     ;Timer #2
        move.b  d1,T2CH(a2)
        move.b  d1,SHR(a2)      ;Shift register
;
        move.b  #$7f,IER(a3)    ;All interrupts off
        move.b  d1,IFR(a3)
        move.b  d1,ACR(a3)      ;Aux control
        move.b  d1,PCR(a3)      ;Per control
        move.b  d1,ORB(a3)      ;Ports
        move.b  d1,ORA(a3)
        move.b  d1,DDRB(a3)     ;Direction
        move.b  d1,DDRA(a3)
        move.b  d1,T1CL(a3)     ;Timer #1
        move.b  d1,T1CH(a3)
        move.b  d1,T1LL(a3)     ;Latch
        move.b  d1,T1LH(a3)
        move.b  d1,T2CL(a3)     ;Timer #2
        move.b  d1,T2CH(a3)
        move.b  d1,SHR(a3)      ;Shift register
        rts
;
;=============================================================================
;
; Check data line connection to board.
;
CHECKLINE
        bsr     INITBOARD
        clr     d1              ; d1 holds data to write to 6522
@1      move.b  d1,T1LL(a2)     ; try writing to the 6522
        move.b  T1LL(a2),d2     ; and reading back
        cmp.b   d1,d2           ; was the data the same?
        beq.s   @2
        cmpi.w  #$10,d5         ;Tech mode, ignore fails?
        beq.s   @2
        move.w  #2,d7           ;Error, data line to test card is bad,or timer
        move.w  #1,d0           ;Set fail flag
        bra.s   @3
;
@2      addq.b  #1,d1           ; go thru all 256 data combinations to check
        bne.s   @1              ;   data line connection to I/O board
        clr.w   d0              ;Set pass flag
@3      rts
;
;
;=============================================================================
;
; Check can get interrupt from I/O Port board.
;
CKINTERRUPT
        bsr     INITBOARD
        ori #$0700,sr         ; mask all interrupts
        move.b  #$C0,IER(a2)    ; enable timer 1 interrupts from parallel-port
        move.b  #$01,T1CL(a2)   ; and set a short (1 usec) timeout interval
        move.b  #$00,T1CH(a2)
        NOP                     ;wait a few microseconds for all the interrupt
        NOP                     ;   to be asserted
        NOP
        NOP
;
; set up new default interrupt routines for timer
        clr.w   d2              ; clear interrupt flag
        move.l  (a4),a1         ;...get vector address
        move.l  a1,SAVEVECTOR(a5) ;...Save old vector
        lea     INTTIMER,a1     ;New interrupt vector
        move.l  a1,(a4)         ;...into vector
```

```
;
        clr.w   d0              ;set pass flag, in case we make it thru
        move    #$2200,sr       ; let the 6522 interrupt come thru
        cmpi.w  #$03,d2         ; did we get the interrupt?
        beq.s   @1              ;...yes, good
        move.w  #3,d7           ;no, error
        move.w  #1,d0           ;set error flag
;
@1      MOVE    #$2700,SR       ;disable all interrupts and restore interrupt
        move.l  SAVEVECTOR(a5),a1
        move.l  a1,(a4)
        move.b  #$40,IER(a2)    ; turn off 6522 timer interrupts
        RTS
;==============================
; the following is interrupt
; handler  for the preceding
; routine.
;==============================
INTTIMER moveq   #$03,d2        ; note the level 3 or up interrupt
        move.b  #$40,IER(a2)    ; disable the source
        rte                     ; and return
;
;=========================================================================
; Check registers on 6522(VIA)
CK1VIA  move.l  a2,a0           ;Get address of first VIA
        bsr     DOCKVIA
        rts
;
; Check registers on 6522(VIA)
CK2VIA  move.l  a3,a0           ;Get address of second VIA
        bsr     DOCKVIA
        rts
;
;
;
DOCKVIA
        bsr     INITBOARD
        lea     AVAILABLE,a1    ;Registers that can be tested
        clr     d3              ;Init counter
;
@1      move.w  (a1),d2         ;See if legal register
        beq     @4              ;...skip if zero
;
        clr.l   d1              ; d1 holds data to write to 6522
@2      move.l  a0,12(a6)       ;debug
        move.b  d1,(a0)         ; try writing to the 6522
        move.b  (a0),d2         ; and reading back
        move.b  d1,18(a6)       ; debug
        move.b  d2,20(a6)       ; debug
        cmp.b   d1,d2           ; was the data the same?
        beq.s   @3
        cmpi.w  #$12,d5         ;Tech mode to ignore fails?
        beq.s   @3
        move.w  #2,d7           ;Error, register wrong
        move.w  #1,d0           ;Set fail flag
        bra.s   @10
;
@3      addq.b  #1,d1           ; go thru all 256 data combinations to check
        bne.s   @2              ;   data line connection to I/O board
        clr.b   (a0)            ;Set to 0 on exit
;
@4      addq.b  #1,d3           ;Increment counter
        adda    #8,a0
        adda    #2,a1           ;Next register
        cmp.w   #13,d3
        bmi     @1
;
        clr.w   d0              ;Set pass flag
@10     rts
;
;=========================================================================
;
;Check timers on 6522(VIA)
CK1TIMER  move.l  a2,a0           ;Get address of first VIA
        bsr     CKTIMER
        rts
;
; Check timers on 6522(VIA)
CK2TIMER  move.l  a3,a0           ;Get address of second VIA
        bsr     CKTIMER
        rts
;
;
CKTIMER
        bsr     INITBOARD
        clr.w   d0
        move.w  d0,WhichTimer(a5) ;Init to timer #1
        move.b  #0,ACR(a0)      ; Set for one-shot interrupt
        move.b  #$C0,IER(a0)    ; Enable Timer
;
;  init interrupt address
        ori.w   #$0700,sr       ;Disable interrupts
        lea     INTCARD,a1
        move.l  (a1),a4
        move.l  (a4),a1         ;...get vector address
        move.l  a1,SAVEVECTOR(a5) ;...Save old vector
        lea     ITIMER,a1           ;New interrupt vector
        move.l  a1,(a4)             ;...into vector
        move    #$2100,sr           ; let the 6522 interrupt come thru
;
@99     lea     EXPECT,a4           ;Expect times
        lea     TESTVALUES,a1
        clr.w   d3                  ;Test value counter
@1      clr.w   T1(a5)              ;clear happened flag
;
        clr.w   d2                  ;Init counter for loops done
        move.w  44(a1),d0           ;Get MSB
        move.w  (a1)+,d4            ;Get LSB
        move.w  WhichTimer(a5),d1
        bne     @100
        move.b  d4,T1CL(a0)         ;Load counter low
        move.b  d0,T1CH(a0)         ;Load upper and start counter
        bra     @2
;
@100    move.b  d4,T2CL(a0)         ;Load counter low
        move.b  d0,T2CH(a0)         ;Load upper and start counter
        bra     @2                  ;this keeps paths equal
;
;!!!! Critical loop, any changes require expect table changes
; this loop measured at about 13us
@2      cmpi.w  #0,T1(a5)           ;check for happened flag         ( 8)
        bne     @3                  ;...exit on flag                 ( 8)
        add.w   #1,d2               ;increment timeout timer         ( 4)
        cmp.w   #$7ff0,d2            ;...exit on overtime            ( 4)
        bpl     @3                  ;                                ( 8)
        move.b  (a0),d0             ;Dummy for timing software loop  ( 8)
        bra     @2                  ;Continue wait                   (12)
;!!!! End of critical loop
;
@3      move.w  d2,(a6)             ;DEBUG save
        move.w  T1(a5),d1
        cmpi.w  #33,d5             ;Tech modes to ignore fails?
        beq     @4
        cmp.w  (a4),d2              ;Compare to low limit
        bmi     @7
        move.w  44(a4),d0           ;Compare to high limit
        cmp.w   d0,d2
        bpl     @8
@4      adda    #2,a6
        adda    #2,a4
;
        add.w   #1,d3               ;Increment test value counter
        cmpi.w  #22,d3
        bmi     @1
        adda    #2,a6
        move.w  WhichTimer(a5),d0
        beq     @10
;
        clr.w   d0
@5      move.b  #$7f,IER(a0)   ;Disable Timer #1
        MOVE    #$2700,SR      ;disable all interrupts and restore interrupt
        lea     INTCARD,a1
        move.l  (a1),a4
        move.l  SAVEVECTOR(a5),a1
        move.l  a1,(a4)
        rts
;
@6      move.w  #1,d0               ;Set error flag
        bra     @5
;
; Timer was too fast, time is too short
@7      move.w  #4,d7
        move.w  #1,d0
        bra     @5
;
; Timer was too slow, time is too long
@8      move.w  #5,d7
        move.w  #1,d0
        bra     @5
;
; ===================
@10     add.w   #1,d6                       ;Next Step number
        move.w  #1,d0
        move.w  d0,WhichTimer(a5)
        move.b  #$7f,IER(a0)                ;Disable Timer #1
        move.b  #$A0,IER(a0)                ;Enable Timer #2
        bra     @99
;
; =================================
;
; Interrupt routine for timer test.
;
ITIMER          movem.l a0-a2/d0,-(sp)  ;save registers on interrupt entry
                move.b  IFR(a0),d0      ;See if timer #1
                btst    #6,d0           ;...timer 1 flag
                beq     @1              ;...not timer if flag zero
                move.b  T1CL(a0),d0     ;reset timer1 interrupt
                move.w  #$11,T1(a5)     ;set happened flag
                bra     @2
;
@1              btst    #5,d0           ;Is it timer #2 interrupt?
                beq     @2              ;...no, ignore any others
                move.b  T2CL(a0),d0     ;Clear interrupt
                move.w  #$22,T1(a5)     ;...yes, set flag to program
;
@2              movem.l (sp)+,a0-a2/d0  ;restore regs on interrupt exit
                rte
;
;
;=============================================================================
PA0to7          ;Check connection thru cable
                bsr     INITBOARD
;
                move.b  #$0C,DDRB(a2)   ;PB 2,3 outputs
                move.b  #$00,ORB(a2)
                move.b  #$0C,DDRB(a3)   ;PB 2,3 outputs
                move.b  #$08,ORB(a3)
                move.b  #$FF,DDRA(a2)   ;All outputs
;
                cmp.w   #10,d5          ;Cycle tech mode?
                beq     @20
                cmp.w   #11,d5          ;Cycle tech mode?
                beq     @1
;
                move.w  #$ff,d3
                move.b  d3,ORA(a2)
                nop
                move.b  ORA(a3),d0
                cmp.b   d3,d0
                bne     @11
;
                move.w  #$00,d3
                move.b  d3,ORA(a2)
                nop
                move.b  ORA(a3),d0
                cmp.b   d3,d0
                bne     @11
;
                move.w  #$55,d3
                move.b  d3,ORA(a2)
                nop
                move.b  ORA(a3),d0
                cmp.b   d3,d0
                bne     @11
;
                move.w  #$AA,d3
                move.b  d3,ORA(a2)
                nop
                move.b  ORA(a3),d0
                cmp.b   d3,d0
                bne     @11
;
@1              clr.w   d0
                move.b  d0,DDRA(a2)
                move.b  d0,DDRB(a2)
                move.b  d0,DDRA(a3)
                move.b  d0,DDRB(a3)
;
                move.b  #$0C,DDRB(a3)   ;PB 2,3 outputs
                move.b  #$00,ORB(a3)
                move.b  #$0C,DDRB(a2)   ;PB 2,3 outputs
                move.b  #$08,ORB(a2)
                move.b  #$FF,DDRA(a3)   ;All outputs
                cmp.w   #11,d5
                beq     @21
;
                move.w  #$ff,d3
                move.b  d3,ORA(a3)
                nop
                move.b  ORA(a2),d0
                cmp.b   d3,d0
                bne     @12
;
                move.w  #$00,d3
                move.b  d3,ORA(a3)
                nop
                move.b  ORA(a2),d0
                cmp.b   d3,d0
                bne     @12
;
                move.w  #$55,d3
                move.b  d3,ORA(a3)
                nop
                move.b  ORA(a2),d0
                cmp.b   d3,d0
                bne     @12
;
                move.w  #$AA,d3
                move.b  d3,ORA(a3)
                nop
                move.b  ORA(a2),d0
                cmp.b   d3,d0
                bne     @12
;
@10             rts
;
@11             move.w  d0,d4       ;Actual, d3 contains expected
                move.w  #6,d7
                bra     @10
;
@12             move.w  d0,d4       ;Actual, d3 contains expected
                move.w  #7,d7
                bra     @10
;
; Tech mode loop, send data from port A to port B
@20             move.b  #$55,ORA(a2)
                move.b  #$AA,ORA(a2)
                bra     @20
;
; Tech mode loop, send data from port B to port A
@21             move.b  #$55,ORA(a3)
                move.b  #$AA,ORA(a3)
                bra     @21
;
;=========================================================================
;
PB0and7         ;Check connection thru cable PB0,PB7
                bsr     INITBOARD
;
                move.b  #$8C,DDRB(a2)   ;PB 2,3,7 output
                move.b  #$0C,DDRB(a3)   ;PB 0 input
                move.b  #$00,ORB(a3)
                cmp.w   #12,d5
                beq     @20
                cmp.w   #13,d5
                beq     @1
;
                move.w  #$80,d3
                move.b  d3,ORB(a2)
                move.w  #$01,d3
                nop
                nop
                move.b  ORB(a3),d0
                and.w   #$0001,d0
                cmp.b   d3,d0
                bne     @11
;
                move.w  #$00,d3
                move.b  d3,ORB(a2)
                nop
                move.b  ORB(a3),d0
                and.w   #$0001,d0
                cmp.b   d3,d0
                bne     @11
;
@1              move.b  #$8C,DDRB(a3)   ;PB 2,3,7 output
                move.b  #$0C,DDRB(a2)   ;PB 0 input
                move.b  #$00,ORB(a2)
                cmp.w   #13,d5
                beq     @21
;
                move.w  #$80,d3
                move.b  d3,ORB(a3)
                move.w  #$01,d3
                nop
                move.b  ORB(a2),d0
                and.w   #$0001,d0
                cmp.b   d3,d0
                bne     @12
;
                move.w  #$00,d3
                move.b  d3,ORB(a3)
                nop
                move.b  ORB(a2),d0
                and.w   #$0001,d0
                cmp.b   d3,d0
                bne     @12
;
@10             rts
;
@11             move.w  d0,d4       ;Actual, d3 contains expected
                move.w  #8,d7
                bra     @10
;
@12             move.w  d0,d4       ;Actual, d3 contains expected
                move.w  #9,d7
                bra     @10
;
; Tech mode loop
@20             move.b  #$80,ORB(a2)
                move.w  #$00,ORB(a2)
                bra     @20
;
; Tech mode loop
@21             move.b  #$80,ORB(a3)
                move.w  #$00,ORB(a3)
                bra     @21
;
;=========================================================================
;
PB4and6         ;Check connection thru cable PB4,PB6
                bsr     INITBOARD
;
                move.b  #$1C,DDRB(a2)   ;PB 2,3,4 output
                move.b  #$0C,DDRB(a3)   ;PB 6 input
                move.b  #$00,ORB(a3)
                cmp.w   #14,d5          ;Tech mode?
                beq     @20
                cmp.w   #15,d5
                beq     @1
;
                move.w  #$10,d3
                move.b  d3,ORB(a2)
                move.w  #$40,d3
                nop
                nop
                move.b  ORB(a3),d0
                and.w   #$0040,d0
                cmp.b   d3,d0
                bne     @11
;
                move.w  #$00,d3
                move.b  d3,ORB(a2)
                nop
                move.b  ORB(a3),d0
                and.w   #$0040,d0
                cmp.b   d3,d0
                bne     @11
;
@1              move.b  #$1C,DDRB(a3)   ;PB 2,3,4 output
                move.b  #$0C,DDRB(a2)   ;PB 6 input
                move.b  #$00,ORB(a2)
                cmp.w   #15,d5
                beq     @21
;
                move.w  #$10,d3
                move.b  d3,ORB(a3)
                move.w  #$40,d3
                nop
                move.b  ORB(a2),d0
                and.w   #$0040,d0
                cmp.b   d3,d0
                bne     @12
;
                move.w  #$00,d3
                move.b  d3,ORB(a3)
                nop
                move.b  ORB(a2),d0
                and.w   #$0040,d0
                cmp.b   d3,d0
                bne     @12
;
@10             rts
;
@11             move.w  d0,d4       ;Actual, d3 contains expected
                move.w  #10,d7
                bra     @10
;
@12             move.w  d0,d4       ;Actual, d3 contains expected
                move.w  #11,d7
                bra     @10
;
; Tech mode
@20             move.b  #$10,ORB(a2)
                move.w  #$00,ORB(a2)
                bra     @20
;
; Tech mode
@21             move.b  #$10,ORB(a3)
                move.w  #$00,ORB(a3)
                bra     @21
;
;=========================================================================
;
PB3andCB1       ;Check connection thru cable PB3,CB1
                bsr     INITBOARD
;
; Setup vector to expect CB1 interrupt
                ori.w   #$0700,sr       ;Disable interrupts
                lea     INTCARD,a1
                move.l  (a1),a4
                move.l  (a4),a1         ;...get vector address
                move.l  a1,SAVEVECTOR(a5) ;...Save old vector
                lea     ICB1V1,a1       ;New interrupt vector
                move.l  a1,(a4)         ;...into vector
                move    #$2100,sr       ; let the 6522 interrupt come thru
;
                move.b  #$0C,DDRB(a2)   ;PB 2,3 output
                move.b  #$00,ORB(a2)    ;make an output gate
                move.b  #$0C,DDRB(a3)   ;CB1 input
                move.b  #$08,ORB(a3)    ;Make an input gate
;
; Set PCR to interrupt on high going edge
                move.b  #$10,PCR(a3)
                move.w  #$00,CB1VIA1(a5) ;clear happened flag
;
; Enable CB1 interrupt capability
                move.b  #$7f,IFR(a3)    ;All flags off
                move.b  #$90,IER(a3)
;
```

```
                cmp.w   #16,d5          ;Tech mode loop?
                beq     @20
                cmp.w   #17,d5          ;Tech mode loop?
                beq     @1
;
; Wait and assure no interrupts
                nop
                nop
                move.w  CB1VIA1(a5),d0
                cmp.w   #0,d0
                bne     @12
;
; Drive a 1 out of PB3
                move.b  #$08,ORB(a2)
                nop
                nop
;
; Check to see if got interrupt, fail if did not.
                move.w  CB1VIA1(a5),d0
                cmp.w   #0,d0
                beq     @13
;
; Reset got interrupt flag
                move.w  #0,CB1VIA1(a5)
;
; Drive a 0 out of PB3
                move.b  #$00,ORB(a2)
                nop
                nop
;
; Check an assure no interrupt
                move.w  CB1VIA1(a5),d0
                cmp.w   #0,d0
                bne     @12
;
; Drive a 1 out of PB3
                move.b  #$08,ORB(a2)
                nop
                nop
;
; Check to see if got interrupt, fail if did not.
                move.w  CB1VIA1(a5),d0
                cmp.w   #0,d0
                beq     @13
;
; Disable CB1 interrupt.
@1              move.b  #$7F,IER(a2)
;
;
; Setup vector to expect CB1 interrupt
                ori.w   #$0700,sr       ;Disable interrupts
                lea     INTCARD,a1
                move.l  (a1),a4
                lea     ICB1V2,a1       ;New interrupt vector
                move.l  a1,(a4)         ;... into vector
                move    #$2100,sr       ; let the 6522 interrupt come thru
;
                move.b  #$0C,DDRB(a3)   ;PB 2,3 output
                move.b  #$00,ORB(a3)    ;make an output gate
                move.b  #$0C,DDRB(a2)   ;CB1 input
                move.b  #$08,ORB(a2)    ;Make an input gate
;
; Set PCR to interrupt on high going edge
                move.b  #$10,PCR(a2)
                move.w  #$00,CB1VIA2(a5) ;clear happened flag
;
; Enable CB1 interrupt capability
                move.b  #$7f,IFR(a2)   ;All flags off
                move.b  #$90,IER(a2)
;
                cmp.w   #17,d5          ;Tech mode loop
                beq     @21
;
; Wait and assure no interrupts
                nop
                nop
                move.w  CB1VIA2(a5),d0
                cmp.w   #0,d0
                bne     @11
;
; Drive a 1 out of PB3
                move.b  #$08,ORB(a3)
                nop
                nop
;
; Check to see if got interrupt, fail if did not.
                move.w  CB1VIA2(a5),d0
                cmp.w   #0,d0
                beq     @14
;
; Reset got interrupt flag
                move.w  #0,CB1VIA2(a5)
;
; Drive a 0 out of PB3
                move.b  #$00,ORB(a3)
                nop
                nop
;
; Check an assure no interrupt
                move.w  CB1VIA2(a5),d0
                cmp.w   #0,d0
                bne     @11
;
; Drive a 1 out of PB3
                move.b  #$08,ORB(a3)
                nop
                nop
;
; Check to see if got interrupt, fail if did not.
                move.w  CB1VIA2(a5),d0
                cmp.w   #0,d0
                beq     @14
;
; Disable CB1 interrupt.
                move.b  #$7F,IER(a3)
;
;
;
;
; Restore vector to original
                MOVE    #$2700,SR  ;disable all interrupts and rest interrupt
                lea     INTCARD,a1
                move.l  (a1),a4
                move.l  SAVEVECTOR(a5),a1
                move.l  a1,(a4)
;
@10             rts
;
;
; Extranous interrupt error on port A
@11             move.w  #12,d7
                bra     @10
;
; Extranous interrupt error on port B
@12             move.w  #13,d7
                bra     @10
;
; Expected interrupt did not occur on port B
@13             move.w  #14,d7
                bra     @10
;
; Expected interrupt did not occur on port A
@14             move.w  #15,d7
                bra     @10
;
;
; Tech mode loop
@20             move.b  #$08,ORB(a2)    ; Drive a 1 out of PB3
                nop
                nop
                move.b  #$00,ORB(a2)    ; Drive a 0 out of PB3
                nop
                nop
                bra     @20
;
; Tech mode loop
@21             move.b  #$08,ORB(a3)    ; Drive a 1 out of PB3
                nop
                nop
                move.b  #$00,ORB(a3)    ; Drive a 0 out of PB3
                nop
                nop
                bra     @21
;
;
; Interrupt routine for PB3 to CB1 test.
;
ICB1V1          move.l  d0,-(sp)        ;save registers on interrupt entry
                move.b  IFR(a3),d0      ;See if CB1
                btst    #4,d0           ;...CB1 flag
                beq     @1              ;...not CB1 if flag zero
                tst.b   ORB(a3)         ;reset CB1 interrupt
                move.w  #$01,CB1VIA1(a5) ;set happened flag
;
@1              move.l  (sp)+,d0        ;restore regs on interrupt exit
                rte
;
; Interrupt routine for PB3 to CB1 test.
;
ICB1V2          move.l  d0,-(sp)        ;save registers on interrupt entry
                move.b  IFR(a2),d0      ;See if CB1
                btst    #4,d0           ;...CB1 flag
                beq     @1              ;...not CB1 if flag zero
                tst.b   ORB(a2)         ;reset CB1 interrupt
                move.w  #$01,CB1VIA2(a5)    ;set happened flag
;
@1              move.l  (sp)+,d0        ;restore regs on interrupt exit
                rte
;
;==========================================================================
CA2andPB1       ;Check connection thru cable CA2,PB1
                bsr     INITBOARD
```

```
;
; Setup vector to expect CA1 interrupt
                ori.w   #$0700,sr       ;Disable interrupts
                lea     INTCARD,a1
                move.l  (a1),a4
                move.l  (a4),a1         ;...get vector address
                move.l  a1,SAVEVECTOR(a5) ;...Save old vector
                lea     ICA1V1,a1       ;New interrupt vector
                move.l  a1,(a4)         ;...into vector
                move    #$2100,sr       ; let the 6522 interrupt come thru
;
                move.b  #$0C,DDRB(a2).  ;PB 2,3 output
                move.b  #$00,ORB(a2)    ;make an output gate
                move.b  #$0C,DDRB(a3)   ;PB1 input
                move.b  #$08,ORB(a3)    ;Make an input gate
;
                move.b  #$0C,PCR(a2)    ;Drive CA2 Low
; Set PCR to interrupt on high going edge
                move.b  #$01,PCR(a3)
                move.w  #$00,CA1VIA1(a5) ;clear happened flag
;
; Enable CB1 interrupt capability
                move.b  #$7f,IFR(a3)    ;All flags off
                move.b  #$82,IER(a3)
;
                cmp.w   #18,d5          ;Tech mode loop?
                beq     @20
                cmp.w   #19,d5
                beq     @1
;
; Wait and assure no interrupts
                nop
                nop
                move.w  CA1VIA1(a5),d0
                cmp.w   #0,d0
                bne     @11
;
                move.b  #$0E,PCR(a2)    ;Drive CA2 High
                nop
                nop
;
; Check to see if got interrupt, fail if did not.
                move.w  CA1VIA1(a5),d0
                cmp.w   #0,d0
                beq     @12
;
; Check PB1 for constant level
                move.b  ORB(a3),d0
                and.b   #$02,d0
                cmp.b   #$02,d0
                bne     @17
;
; Reset got interrupt flag
                move.w  #0,CA1VIA1(a5)
;
                move.b  #$0C,PCR(a2)    ;Drive CA2 Low
                nop
                nop
;
; Check an assure no interrupt
                move.w  CA1VIA1(a5),d0
                cmp.w   #0,d0
                bne     @11
;
; Check PB1 for constant level
                move.b  ORB(a3),d0
                and.b   #$02,d0
                cmp.b   #$00,d0
                bne     @17
;
                move.b  #$0E,PCR(a2)    ;Drive CA2 High
                nop
                nop
;
; Check to see if got interrupt, fail if did not.
                move.w  CA1VIA1(a5),d0
                cmp.w   #0,d0
                beq     @12
;
; Check PB1 for constant level
                move.b  ORB(a3),d0
                and.b   #$02,d0
                cmp.b   #$02,d0
                bne     @17
;
; Disable CB1 interrupt.
@1              move.b  #$7F,IER(a2)
;
;
; Setup vector to expect CA1 interrupt
                ori.w   #$0700,sr       ;Disable interrupts
                lea     INTCARD,a1
                move.l  (a1),a4
                lea     ICA1V2,a1       ;New interrupt vector
                move.l  a1,(a4)         ;...into vector
                move    #$2100,sr       ; let the 6522 interrupt come thru
```

```
;
                move.b  #$0C,DDRB(a3)   ;PB 2,3 output
                move.b  #$00,ORB(a3)    ;make an output gate
                move.b  #$0C,DDRB(a2)   ;PB1 input
                move.b  #$08,ORB(a2)    ;Make an input gate
;
                move.b  #$0C,PCR(a3)    ;Drive CA2 Low
; Set PCR to interrupt on high going edge
                move.b  #$01,PCR(a2)
                move.w  #$00,CA1VIA2(a5) ;clear happened flag
;
; Enable CB1 interrupt capability
                move.b  #$7f,IFR(a2)   ;All flags off
                move.b  #$82,IER(a2)
;
                cmp.w   #19,d5          ;Tech mode loop?
                beq     @21
;
; Wait and assure no interrupts
                nop
                nop
                move.w  CA1VIA2(a5),d0
                cmp.w   #0,d0
                bne     @14
;
                move.b  #$0E,PCR(a3)    ;Drive CA2 High
                nop
                nop
;
; Check to see if got interrupt, fail if did not.
                move.w  CA1VIA2(a5),d0
                cmp.w   #0,d0
                beq     @13
;
; Check PB1 for constant level
                move.b  ORB(a2),d0
                and.b   #$02,d0
                cmp.b   #$02,d0
                bne     @18
;
; Reset got interrupt flag
                move.w  #0,CA1VIA2(a5)
;
                move.b  #$0C,PCR(a3)    ;Drive CA2 Low
                nop
                nop
;
; Check an assure no interrupt
                move.w  CA1VIA2(a5),d0
                cmp.w   #0,d0
                bne     @14
;
; Check PB1 for constant level
                move.b  ORB(a2),d0
                and.b   #$02,d0
                cmp.b   #$00,d0
                bne     @18
;
                move.b  #$0E,PCR(a3)    ;Drive CA2 High
                nop
                nop
;
; Check to see if got interrupt, fail if did not.
                move.w  CA1VIA2(a5),d0
                cmp.w   #0,d0
                beq     @13
;
; Check PB1 for constant level
                move.b  ORB(a2),d0
                and.b   #$02,d0
                cmp.b   #$02,d0
                bne     @18
;
; Disable CB1 interrupt.
                move.b  #$7F,IER(a3)
;
;
;
;
;
; Restore vector to original
                MOVE    #$2700,SR  ;disable all interrupts and rest interrupt
                lea     INTCARD,a1
                move.l  (a1),a4
                move.l  SAVEVECTOR(a5),a1
                move.l  a1,(a4)
;
@10             rts
;
;
; Interrupt occurred when none expected, on Port B.
@11             move.w  #13,d7
                bra     @10
;
; Expected interrupt did not occur on port B.
@12             move.w  #14,d7
```

```
                bra     @10
;
; Expected interrupt did not occur on port A.
@13             move.w  #15,d7
                bra     @10

;
; Interrupt occurred when none expected, on Port A.
@14             move.w  #12,d7
                bra     @10

;
; PB1 on port B failure (BSY)
@17             move.w  #18,d7
                bra     @10

;
; PB1 on port A failure (BSY)
@18             move.w  #19,d7
                bra     @10

;
; Tech mode loop
@20             move.b  #$0E,PCR(a2)    ;Drive CA2 High
                nop
                nop
                move.b  #$0C,PCR(a2)    ;Drive CA2 Low
                nop
                nop
                bra     @20
;
; Tech mode loop
@21             move.b  #$0E,PCR(a3)    ;Drive CA2 High
                nop
                nop
                move.b  #$0C,PCR(a3)    ;Drive CA2 Low
                nop
                nop
                bra     @21
;
;
; Interrupt routine for PB3 to CA1 test.
;
ICA1V1          move.l  d0,-(sp)        ;save registers on interrupt entry
                move.b  IFR(a3),d0      ;See if CA1
                btst    #1,d0           ;...CA1 flag
                beq     @1              ;...not CA1 if flag zero
                tst.b   ORA(a3)         ;reset CA1 interrupt
                move.w  #$01,CA1VIA1(a5) ;set happened flag
;
@1              move.l  (sp)+,d0        ;restore regs on interrupt exit
                rte
;
; Interrupt routine for PB3 to CA1 test.
;
ICA1V2          move.l  d0,-(sp)        ;save registers on interrupt entry
                move.b  IFR(a2),d0      ;See if CA1
                btst    #1,d0           ;...CA1 flag
                beq     @1              ;...not CA1 if flag zero
                tst.b   ORA(a2)         ;reset CA1 interrupt
                move.w  #$01,CA1VIA2(a5)     ;set happened flag
;
@1              move.l  (sp)+,d0        ;restore regs on interrupt exit
                rte
;
;==========================================================================
;
;   a2 - First VIA address.
;   a3 - Second VIA address.
AParity
;                                        DDRA set to inputs, A port
                bsr     INITBOARD
;
                move.b  #$FF,DDRA(a3)   ;Make port B output data
                move.b  #$1C,DDRB(a3)   ;PB 2,3,4 output
                move.b  #$00,ORB(a3)    ;Make an output gate
;
                move.b  #$2C,DDRB(a2)   ;PB 2,3,5 output
                move.b  #$08,ORB(a2)    ;make an input gate
                move.b  #$6A,PCR(a2)    ;CA2 pulse output, CB2 pos edge ind inter
;
                cmp.w   #20,d5          ;Tech mode loop?
                beq     @20
;
                lea     PData,a0
                lea     PExpect,a1
                clr.w   ParityTests(a5) ;Clear test number-counter
@1
                move.w  (a0),d0
                move.b  d0,ORA(a3)      ;Data
                move.w  44(a0),d1
                adda    #2,a0
                move.b  d1,ORB(a3)      ;Parity line
                move.b  #$08,ORB(a2)    ;Pulse reset line high
                move.b  #$28,ORB(a2)    ;      reset line low
                move.b  #$08,IFR(a2)    ;Reset CB2 flag in case it was set
                move.b  ORA(a2),d3      ;Read port A
                cmp.b   d0,d3           ;...assure data is correct
                bne     @11
                clr.w   d4
                move.b  IFR(a2),d4       ;Read see if CB2 was flagged.
                move.w  d4,44(a1)        ;...save actual
                move.w  (a1)+,d1
                eor.w   d1,d4            ;Only leave CB2 difference
                and.w   #$08,d4
                cmp.w   #0,d4
                bne     @12
;
                move.w  ParityTests(a5),d0
                add.w   #1,d0
                move.w  d0,ParityTests(a5)
                cmp.w   #22,d0
                bne     @1
;
; End of loop
@10             rts
;
; Bad data (PA0-PA7) during parity test.
@11             move.w  #20,d7
                bra     @10
;
; Bad parity (CB2) during parity test.
@12             move.w  #21,d7
                bra     @10
;
;Tech Mode loop
@20             move.b  #$01,ORA(a3)     ;Data
                move.b  #$01,ORB(a3)     ;Parity line
                move.b  #$08,ORB(a2)     ;Pulse reset line high
                move.b  #$28,ORB(a2)     ;      reset line low
                move.b  #$08,IFR(a2)     ;Reset CB2 flag in case it was set
                move.b  ORA(a2),d3       ;Read port A
                move.b  IFR(a2),d4       ;Read see if CB2 was flagged.
;
                move.b  #$03,ORA(a3)     ;Data
                move.b  #$01,ORB(a3)     ;Parity line
                move.b  #$08,ORB(a2)     ;Pulse reset line high
                move.b  #$28,ORB(a2)     ;      reset line low
                move.b  #$08,IFR(a2)     ;Reset CB2 flag in case it was set
                move.b  ORA(a2),d3       ;Read port A
                move.b  IFR(a2),d4       ;Read see if CB2 was flagged.
                bra     @20
;
;==================================================================
;
BParity
;                                        DDRA set to inputs, A port
                bsr     INITBOARD
;
                move.b  #$FF,DDRA(a2)    ;Make port B output data
                move.b  #$1C,DDRB(a2)    ;PB 2,3,4 output
                move.b  #$00,ORB(a2)     ;Make an output gate
;
                move.b  #$2C,DDRB(a3)    ;PB 2,3,5 output
                move.b  #$08,ORB(a3)     ;make an input gate
                move.b  #$6A,PCR(a3)     ;CA2 pulse output, CB2 pos edge ind inter
;
                cmp.w   #21,d5
                beq     @20
;
                lea     PData,a0
                lea     PExpect,a1
                clr.w   ParityTests(a5) ;Clear test number counter
@1
                move.w  (a0),d0
                move.b  d0,ORA(a2)       ;Data
                move.w  44(a0),d1
                adda    #2,a0
                move.b  d1,ORB(a2)       ;Parity line
                move.b  #$08,ORB(a3)     ;Pulse reset line high
                move.b  #$28,ORB(a3)     ;      reset line low
                move.b  #$08,IFR(a3)     ;Reset CB2 flag in case it was set
                move.b  ORA(a3),d3       ;Read port A
                cmp.b   d0,d3            ;...assure data is correct
                bne     @11
                clr.w   d4
                move.b  IFR(a3),d4       ;Read see if CB2 was flagged.
                move.w  d4,44(a1)        ;...save actual
                move.w  (a1)+,d1
                eor.w   d1,d4            ;Only leave CB2 difference
                and.w   #$08,d4
                cmp.w   #0,d4
                bne     @12
;
                move.w  ParityTests(a5),d0
                add.w   #1,d0
                move.w  d0,ParityTests(a5)
                cmp.w   #22,d0
                bne     @1
;
; End of loop
@10             rts
;
; Bad data (PA0-PA7) during parity test.
@11             move.w  #22,d7
                bra     @10
```

```
;
; Bad parity (CB2) during parity test.
@12             move.w  #23,d7
                bra     @10
;
;
;Tech Mode loop
@20             move.b  #$01,ORA(a2)    ;Data
                move.b  #$01,ORB(a2)    ;Parity line
                move.b  #$08,ORB(a3)    ;Pulse reset line high
                move.b  #$28,ORB(a3)    ;       reset line low
                move.b  #$08,IFR(a3)    ;Reset CB2 flag in case it was set
                move.b  ORA(a3),d3      ;Read port A
                move.b  IFR(a3),d4      ;Read see if CB2 was flagged.
;
                move.b  #$03,ORA(a2)    ;Data
                move.b  #$01,ORB(a2)    ;Parity line
                move.b  #$08,ORB(a3)    ;Pulse reset line high
                move.b  #$28,ORB(a3)    ;       reset line low
                move.b  #$08,IFR(a3)    ;Reset CB2 flag in case it was set
                move.b  ORA(a3),d3      ;Read port A
                move.b  IFR(a3),d4      ;Read see if CB2 was flagged.
                bra     @20
;
;====================================================================
;
; Restore VIA for next user
;
RESTORE
                move    #$2700,sr       ;Disable interrupts
;
                lea     KBDVIA,a1       ;Enable keyboard COPS
                move.b  #$7f,IER(a1)
                move.b  KBDIEN(a5),d0   ;Get from LOCAL table
                move.b  d0,IER(a1)
;
                clr.b   ACR(a2)
                move.b  #$7f,IFR(a2)
                move.b  #$7f,IER(a2)    ;All interrupts off
                clr.b   ACR(a3)
                move.b  #$7f,IFR(a3)
                move.b  #$7f,IER(a3)    ;All interrupts off
                rts
;
;
;____________________________________________________________________
;____________________________________________________________________
;
CARD            .WORD   0,0     ;Card base address, where boot ROM is
INTCARD         .WORD   0,0     ;Int vector for card
;
LOCAL           .WORD   0       ;TICKTOCK
                .WORD   0       ;T1
                .WORD   0       ;T2
                .WORD   0       ;KBDIEN
                .WORD   0,0     ;SAVEVECTOR
                .WORD   0       ;WhichTimer
                .WORD   0       ;CA1VIA1
                .WORD   0       ;CA1VIA2
                .WORD   0       ;CB1VIA1
                .WORD   0       ;CB1VIA2
                .WORD   0       ;ParityTests
;
; Tables derived from National LOGIC databook for LS280, P6-204
PData           .WORD   $00                     ;Data for PA0-PA7
                .WORD   $01,$02,$04,$07
                .WORD   $03,$06,$05
                .WORD   $08,$10,$20,$38
                .WORD   $18,$30,$28
                .WORD   $40,$80,$00,$C0
                .WORD   $C0,$80,$40
                .WORD   $00                     ;Parity sent, PB6
                .WORD   $00,$00,$00,$00
                .WORD   $00,$00,$00
                .WORD   $00,$00,$00,$00
                .WORD   $00,$00,$00
                .WORD   $00,$00,$10,$10
                .WORD   $00,$10,$10
; Expect $08 = expect CB2 interrupt, 0 = no interrupt expected
PExpect         .WORD   $08                     ;Expected
                .WORD   $00,$00,$00,$00
                .WORD   $08,$08,$08
                .WORD   $00,$00,$00,$00
                .WORD   $08,$08,$08
                .WORD   $00,$00,$00,$00
                .WORD   $08,$08,$08
                .WORD   $00                     ;Actual
                .WORD   $00,$00,$00,$00
                .WORD   $00,$00,$00
                .WORD   $00,$00,$00,$00
                .WORD   $00,$00,$00
                .WORD   $00,$00,$00,$00
                .WORD   $00,$00,$00
;
; Flag for table end and debug
                .WORD   $1234,$5678
```

```
;
;
; Timer test values LSB
TESTVALUES     .WORD   1,   2,   4,   8,  16,   32,   64,  128,   15,  240,  255
               .WORD   0,   0,   0,   0,   0,    0,    0,    0,    0,    0,    0
; Timer test values MSB
               .WORD   0,   0,   0,   0,   0,    0,    0,    0,    0,    0,    0
               .WORD   1,   2,   4,   8,  16,   32,   64,  128,   15,  240,  255
;
EXPECT         .WORD   0,   0,   0,   0,   0,    1,    3,    6,    0,  $0c,  $0d
               .WORD $0d,$1a, $35, $6b, $d7, $1af, $35e, $6bc,  $ca, $ca1, $d6b
;
               .WORD   2,   2,   2,   2,   2,    3,    5,    8,    2,  $0e,  $0f
               .WORD $0f,$1c, $37, $6d, $d9, $1b1, $360, $6be,  $cc, $ca3, $d6d
;
; Registers to test in VIA
; ...DDRB,DDRA,T1LL,T1LH,SR,ACR,PCR
AVAILABLE .WORD 0,0,1,1,0,0,1,1,0,0,1,1,1,0,0,0
;
;
                .WORD    $1234,$5678
DEBUG           .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .WORD    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
;
```

```
%
a+C
TWOP.HDR

TOPORT.LT
■
%%
```