Q64 ERROR CODES, BOARD BUG DIAGNOTICS, AND EXTERNAL SPECS. FOR Q64 DIAGNOSTICS

Q64 ERROR CODES

Start Up Sequence Codes	
00	Start of Diagnostic (IPL light is lit) Micro flag test running
01	ALU & AUX test pass (non-conclusive) Stop switch stuck on or Depressed
02	Stop switch ok, not on (start light on)-test passed Power fail or IPL warning occurring
03	No power fail or IPL warning-test passed Base register test running Address register test running
04	Memory address register and Base register test passed Memory test for power on (cold start only) running
05	Board set passed start-up sequence (IPL light is turned off)
FF (7F)	Clock or pipeline problem

Status after IPL should be 0005

Failure Table

If the following start-up sequence is constant on the diagnostic panel, use the table for troubleshooting.

Codes displayed on panel	ALU	AUX	MADD	MDAT	MEM
00	X	X			
01	X	X			
02		X			
03			X	X	
04			X	X	X
FF (7F)	X	X			

* 05 System checkd ok - no errors

System Level

E0	No device 0 detected (halt light on) → controller 0
E1	Power fail detected
E2	IPL warning detected
E3	Seek to non disk device
B0	Serial Port in use
B1	Serial Port used
BB	Breakpoint set

System Level
To Screen

Appears on TØØ

E4	* "MEM PAR ERR"	This will also display the physical address of the error
E5	* "INDIRECT"	16 levels of indirecting has occurred
E6	* "ILLEGAL"	An illegal instruction has occurred
E7	* "HARDWARE ERROR"	An illegal condition has occurred
E8	* "FETCH ERROR"	An illegal fetch vector occurred ← check MDB board

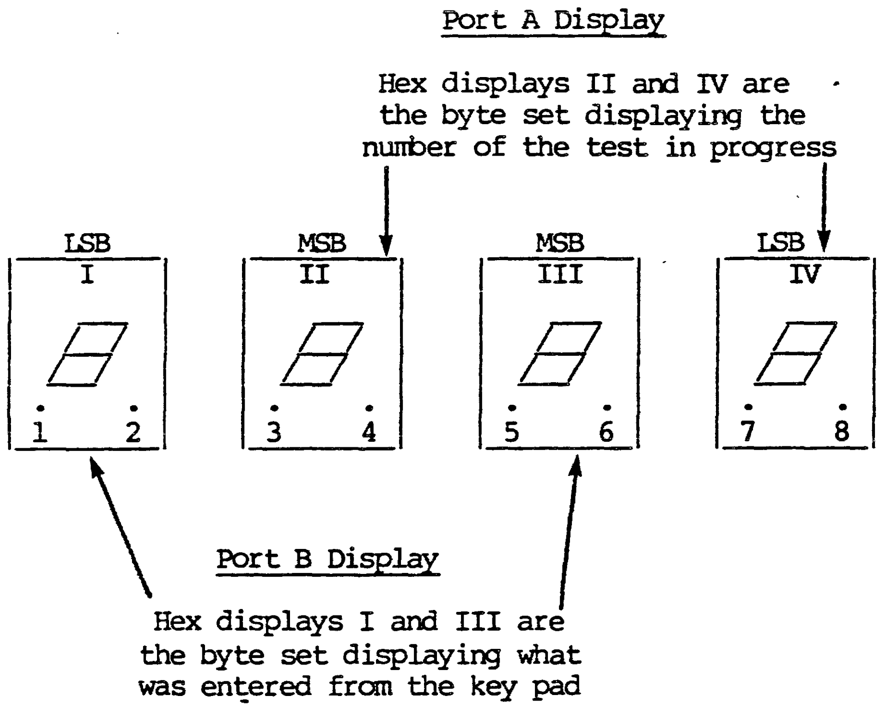
These Error Codes also display the address of the error.

* The only escape for these errors is to IPL

Q64 DIAGNOSTIC PANEL OPERATION

<u>An improved version of this panel will be released in the near future.</u>

Hex Display Operations


Port A Display
Hex displays II and IV are the byte set displaying the number of the test in progress
LSB
I
MSB
II
MSB
III
LSB
IV
1
2
3
4
5
6
7
8
Port B Display
Hex displays I and III are the byte set displaying what was entered from the key pad

---

Decimal Points: the decimal points, when illuminated, indicate various states of the machine

1 MINT = micro interrupt

2 HELP = macro interrupt, usually on

3 = not used

4 = not used

5 CONT = clocks are running

6 HALT = clocks halted

7 PARR = parity error, usually on (off indicates error)

8 STOP = executions stopped (Start/Stop)

If the "B" display is blanked out this will indiate that clocks are not running. AUX board is possible failure.

"B" set of display will display what was entered from the key pad, shifting LSB to MSB and new entry to the LSB

*(Key pad operations are currently supported only under the Q64 diagnostic ROM set.)

Switches: Not currently supported

SW1 = CONT

The CONT will restart the clocks after the HALT mode

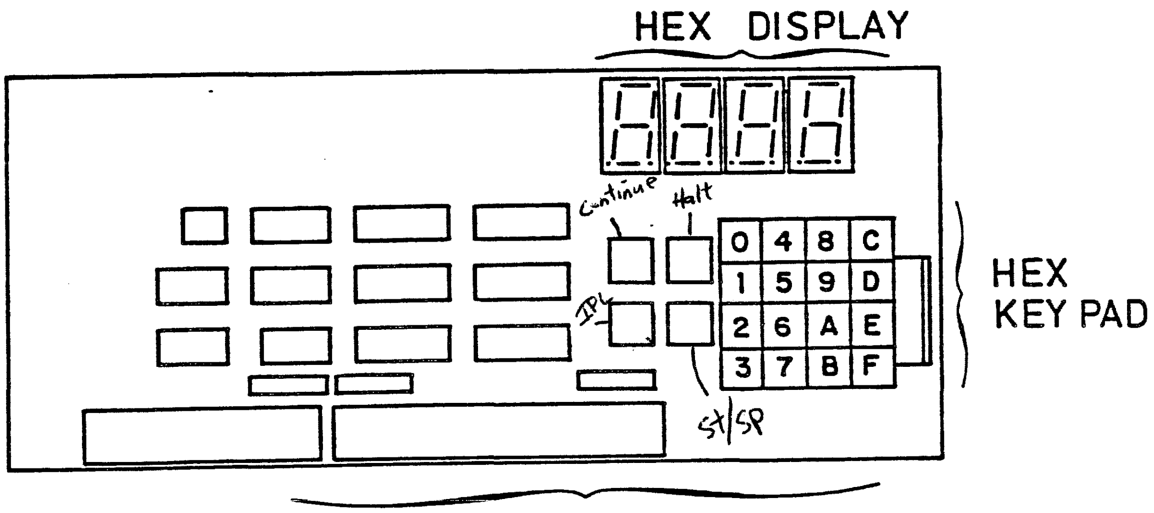
SW2 = HALT

The HALT will hold clock functions - it freezes the state of the machine (it is not a Start/Stop).

SW3 = IPL (initial program load) - same IPL as key switch

SW4 = Start/Stop - stop macro emulation - same as control pod Start/Stop

Key Pad: the keyboard is set up a matrix as per diagram



Q64 DIAGNOSTIC PANEL

Q64 BOARD BUG DIAGNOSTIC ROMS OPERATIONS

The Q64 diagnostic ROM set is used in lieu of the real macro emulation ROM set, to test individual circuits of the Q64 CPU board set.

The purpose of these ROMs is to:

1)	Field:	to aid in fault isolation to a signal board
2)	In-House:	to aid technicians in repair

Each test has a code number it will send out to the diagnostic panel hex display at the start of each test. Each time a test passes, the display will be incremented and the next test run.

The test number entered OR'ed with hex $80, entered on keypad, is echoed on one set of hex display. The other hex display, under CPU control, displays the test number. If the test failed, the test number OR'ed with hex $80 is displayed.

The diagnostic set has 5 modes of operation:

Mode 1) Sequential: (mentioned above) increments through each test sequentially. Sequential mode is entered by placing $00 on the display and then turning the IPL key

Mode 2) Signal Test Execution Mode:

this mode will loop on a signal test

to run in this mode the $80 bit is set, $40 to $00 bits are the test number, e.g. test $15 would be $95. This value is entered on key pad and turn the IPL key.

Mode 3) Halt on Error Mode Freezes State of Machine:

runs in mode 1 order but will halt the clocks on a test that an error occurred on. To run this mode the $40 bit is set (enter a $40 on key pad) and turn IPL key.

Mode 4) Loop:

runs in mode 1 order and stops execution after one complete pass of all tests. If error occurs then the test number OR'ed with the $80 bit will be displayed on the hex display. If all is good, then a $7F will be displayed. To run this mode the $20 bit is set (enter a $20 in key pad) and IPL.

Mode 5) Modes 3 and 4 may be combined for a loop till error and halt (enter $60 on key pad) and IPL.

For test descriptions and running sequences see "BOARD BUG DIAGNOSTIC CODES" switch definitions

The diagnostic panel is used to enter test numbers. Reference "Q64 DIAGNOSTIC PANEL OPERATION".

BOARD BUG DIAGNOSTIC CODES

The following is a list of the tests performed during the BRDBUG diagnostic test and the codes which will be placed out to the front diagnostic panel at the start.

TEST NUMBER	TEST DESCRIPTION	PRIMARY BOARDS TESTED ALU	AUX	MDB	MAD	MEM
00	Kernal Test (Stack, ALU's, Address Bus)	X	X			
01	CPU Live Flag Test	X	X			
02	Shift Mode 0 Test	X	X			
03	Shift Mode 1 Test	X	X			
04	Shift Mode 2 Test	X	X			
05	Shift Mode 3 Test	X	X			
06	General Purpose Counter Zero Test		X			
07	4 Bit GPC Test		X			
08	8 Bit GPC Test		X			
09	12 Bit GPC Test		X			
0A	16 Bit GPC Test		X			
0B	4 Bit GPC Flag Test		X			
0C	4-Way (GPC1 and GPC2) Test		X			
0D	4-Way (GPC1 and GPC2) Test		X			
0E	FFFF and THREE Test		X			
0F	Swap Register Test		X			
10	Local Control Register Test		X			
11	Programmable Interval Timer Test		X	X		
12	Pit Readback Test		X	X		
13	Pit Interrupt Test		X	X		
14	Branch Register Lower Test		X			
15	Branch Register Upper Test		X			
16	Y Branch Lower Test		X			
17	Y Branch Upper Test		X			
18	Branch Register Test (256-Way)		X			
19	Y Branch Test (256-Way)		X			
1A	BCD Zone Test		X			
1B	BCD Adder Test		X			
1C	BCD Subtracter Test		X			
1D	Global Interrupt/Micro Interrupt Test		X			
1E	"Real" Flag Test			X		
1F	Base Register Data Test			X	X	
20	Base Register Address Test			X	X	
21	Logical Address Register A Test			X		
22	Logical Address Register B Test			X		
23	Logical Address Register P Test			X		
24	Real Address Register/Parity Latch Test			X	X	
25	RAR/Perr Upper 8 Bit Test			X	X	
26	Physical Address Register P Test			X	X	
27	Physical Address Register A Test			X	X	
28	Physical Address Register B Test			X	X	
29	Fetch Register Test			X		
2A	Code 1L 16-Way Branch Test			X		

		PRIMARY BOARDS TESTED				
TEST NUMBER	TEST DESCRIPTION	A L U	A U X	M D B	M A D	M E M
2B	Memory Data Bus Test (to array 0)			X	X	X
2C	Memory Bit Test			X	X	X
2D	Memory Address Test (address 0 - 7)			X	X	X
2E	Logical Address Register A Memory Check			X	X	X
2F	Logical Address Register B Memory Check			X	X	X
30	Logical Address Register P Memory Check			X	X	X
31	A Plus 1 Incrementer/8 Byte Write Test				X	X
32	Barrel Shifter Test			X		
33	Lara Increment/Decrement Test			X	X	
34	Larb Increment/Decrement Test			X	X	
35	Larp Increment/Decrement Test			X	X	
36	Base/Bank Hazard Test		X		X	
37	Cascade Register Test			X		
38	Fetch Register/Shifter Test			X		
39	"REAL" Branch Test			X	X	
3A	Indirect Overflow Test			X	X	
3B	Hazard On Instruction Test			X	X	
3C	Base Fault on Indirect Resolution Test			X	X	
3D	Single Operand Instruction Test 1			X	X	
3E	Single Operand Instruction Test 2			X	X	
3F	Double Operand Fetcher Test (no indirects)			X	X	
40	Double Operand Fetcher Test (A indirect)			X	X	
41	Double Operand Fetcher Test (B indirect)			X	X	
42	Double Operand Fetcher Test (A/B indirect)			X	X	
43	Triple Operand Fetcher Test (all cases)			X	X	
44	Fetcher $000F to LARA			X	X	
**46	Memory Test			X	X	X
7F	All Finished With Test(s) or Illegal Test Number					
FF	Stack Error During Kernal Test	X	X			

Any other test number is invalid

** This test is not run during the normal execution of the diagnostic. It can only be accessed by entering $C6 on the panel

This test requires an IOU-39Q set at address 0 and a VT3 be installed with the system.

If an error occurs, the test number 'ORED' with 80 will be put out to the diagnostic panels hex display.

3 9
[illegible]
[illegible]

SWITCH DEFINITIONS

HEX	BITS 7	6	5	4	3	2	1	0	LOOP ON A TEST	HALT ON ERROR	LOOP AT END	
00-1F	0	0	0	X	X	X	X	X	NO	NO	YES	Mode 1
20-3F	0	0	1	X	X	X	X	X	NO	NO	NO	Mode 3
40-5F	0	1	0	X	X	X	X	X	NO	YES	YES	Mode 4
60-7F	0	1	1	X	X	X	X	X	NO	YES	NO	Mode 5
80-FF	1	T	T	T	T	T	T	T	YES	NO	XX	Mode 2

DISPLAY DEFINITIONS

HEX	BITS 7	6	5	4	3	2	1	0	
00-7F	0	X	X	X	X	X	X	X	Test Passed
80-FF	1	T	T	T	T	T	T	T	Test Failed (Bit 7 On) Bits 6-0 define test no. that failed.

NOTE:
T = test number
X = don't care

NOTES, UNLESS OTHERWISE SPECIFIED

REV.	DESCRIPTION	DRFT / DATE	CHK / DATE	PROJ / DATE	MFG / DATE
I	PPR-0382-22	[illegible] 3/14/83	[illegible] 3/14/83	[illegible] 3-14-83	[illegible] 3/14/83
A	PR-0382-27	LUM 4-28-83	—	—	—

S. M.

MAY 31 1983

"PRODUCTION RELEASE"

PROPRIETARY

CE TO PERSONS RECEIVING THIS WING: MDS—QANTEL, INC. MS PROPRIETARY RIGHTS TO ALL RMATION CONTAINED HEREIN. HER THIS DRAWING, NOR ANY ODUCTIONS THEREOF, MAY BE OR DISCLOSED TO ANY THIRD IES WITHOUT THE EXPRESS TEN PERMISSION OF MDS- INC.

NEXT ASSEMBLY A/M44079

APPROVALS		DATE
DRFTS	D. Boal	3/14/83
CHK	[illegible]	4/23/83
PROJ ENGR	[illegible]	5/16/83
MFG ENGR	[illegible]	5-26-83
PROJ MGR	[illegible]	5-25-83

DO NOT SCALE DRAWING

MDS QANTEL BUSINESS COMPUTERS

a Mohawk Data Sciences Company

EXTERNAL SPECIFICATION

Q64 DIAGNOSTIC (BRDDBG)

SIZE	DWG NO.	REV.
A	52084-001	A

SCALE — SHEET 1 OF 16

1.0 SCOPE

This document is intended to provide the user with an understanding of the micro-diagnostics that are used to bring a Q64 CPU up to the point where a macro-diagnostic may be loaded.

2.0 INTRODUCTION AND GENERAL DESCRIPTION

2.1 The diagnostics currently consists of several modules. Each with the intent of debugging several main and where possible, distinct areas of the cpu.
These areas are as follows:
1) The ALU and AUX Boards.
2) The MDAT and MADD Boards.
3) The main memory modules.

2.2 This diagnostic will be resident in the ROM space on the board Since a minimum of hardware need be operational in order to access this. The diagnostic currently fills up about all of the 8k address space. User communication will be via several paths. The primary mode of communication will be done with a logic analyzer connected to the .crostore Address Bus and the "Y" bus. Secondary communication will be done through the "Halt" logic and through the hex displays on the front panel.

2.3 Communication from the user will occur via the front panel switches and to a very small extent, through the " Cont " logic.

2.4 The diagnostic is intended to debug a machine from a working hardcore state to a point where higher level diagnostics may be loaded. The goal is to be able to isolate faults down to the circuit and if possible, the node level.

2.5 This diagnostic starts out by verifing that a small kernel of the machine is operational. As the diagnostic progresses, the kernel will enlarge thus allowing for more specific pin-pointing of faults.

3.0 HARDWARE REQUIREMENTS

3.1 There are two approaches to consider:

1) To debug with the aid of a Rom Simulator.
2) To debug with the aid of diagnostic roms.

3.2 Boths methods will achieve the same results,

Method 1) allows for greater flexibility to the user however it is also at a greater cost in terms of hardware requirements and space requirements.

Method 2) does not give the user as much flexibility , however , a great deal less must be expended as far as equipment and space are concerned.

3.3

ROM SIMULATOR	DIAGNOSTIC ROMS
DVM	DVM
CPU EXTENDER BOARD	CPU EXTENDER BOARD
OSCILLOSCOPE	OSCILLOSCOPE
LOGIC ANALYZER	LOGIC ANALYZER
7.5 OR Q29 SYSTEM:	
ROM SIMULATOR	
48K MAIN MEMORY (min)	
VIDEO TERMINAL	
PRINTER	
ROM RECEIVER/DRIVER BOARD	

4.0 OPERATING INSTRUCTIONS

4.1.1 ROM SIMULATOR METHOD
- a) install rom simulator connections to the Rom Rec/Drv board.
- b) install jumper at (SWEXT) to disable roms.
- c) logic analyzer connections should be made to the MSA bus and the "Y" bus. The clock should be connected to the "TOH" .
- d) From the Q29 system, run Q64LOAD.
  1) enter the hex address for the Rom Simulator.
  2) enter the hex address for the Serial Port communications device (not used in the diagnostic)
  3) a menu should now appear,

     for the diagnostic enter : LW BRDDBG
- e) When the F2 TO CONTINUE prompt comes up , press F2 on the keyboard and IPL on the Q64.

4.1.1 At this point the diagnostic is running, the number for the last completed test will be displayed to the front panel if it is installed. A lookup table or a source listing will be required at this time to find out where the cpu has failed. The logic analyzer can be set to trigger on the trap address to determine the machine history leading up to this event.

4.2.0 The "Diagnostic Rom " method
- a) install the "Roms" on the Cpu.
- b) install the SWEXT jumper.
- c) power the CPU up and press IPL.

4.2.1 at this point the diagnostic is running, the DIAGNOSTIC panel will display the number of the last completed test. Also the lights on the Rom board should be changing at a very fast rate, a stable state on the lights will indicate a trap caused by an error. The data on the lights will directly correspond to the address currently on the MSA bus (similar to the logic analyzer)

4.3 The Diagnostic Panel switches will also define several modes in which the diagnostic will run in. The modes that are currently defined are as follows:

```
Default, no switches pressed ..LOOP AT END
SWITCH $20          ....HALT ON ERROR
                On = halt on error.... Off = Loop on error
SWITCH $80          ....LOOP ON TEST
                On = loop on a test    Off = don't loop
                 other switches = test number
SWITCH $40          ....Halt at end
                 micro program will halt when complete
```

5.0 TEST DESCRIPTIONS

5.1 BRDBUG

5.1.1 This is the first test that should be run .There are several things that should be known about the program.

1) Throughout the program, a test number is displayed to the front panel. Normally this register is incremented as the program progresses.
2) The front panel will be read to determine what paths will be taken during the execution of the test.
3) Register 2 will be the location of the halt command to the local control register. This will be used in the halt on error mode.
4) It is assumed that the machine can reset to MSA $0000
   The main clocks are running (T0H,T3H)
   The "D" source LITERAL decode and Y destination LOCAL CONTROL decode is operational.
   The Z16 flag out of the alu is connected to the two-way branch logic.

5.1.2 KERNAL TEST 1

This will verify that a portion of the literal register and a portion of the Local Control Register are functional. Also ,this will tell us that the HALT logic is working.

5.1.3 CONFIDENCE HALT 2

This will tell us that the Continue logic is working, the INCREMENT the microstore address on literal, and that some jump logic is working.

5.1.4 CONFIDENCE HALT 3

This will happen after Z16 logic has been tested in both states. At this point in the diagnostic, we have a mechanism which gives us the ability to test the rest of the machine.

5.1.5 JUMP TEST

This test will check the independence of the Microstore address lines by doing unconditional jumps through memory and back again. There are two failure paths on this test, the first is to jump out to a wrong location, this can only be tested with a logic analyzer. The second is to skip a jump , this is tested by incrementing a register at each jump (jump counter) and testing it upon returning to the main routine.

5.1.6 LITERAL REGISTER TEST

This is a quick test to check the independence of the bits in the literal register. Unfortunately, since we don't have a great deal of resources available to use at this time, we have to use it to "check itself" this is done by loading the literal register into Register 0 and reloading the literal register and adding it to Register 0 looking for a result of 0.
$AAAA + $5555 + 1 = $0000

5.1.7 ALU FUNCTIONAL TEST

At this point we load Register 1 and Q via the literal register. We then will test the add, subtract, or, and, complement, xor functions on the alu. This will verify that the 2901 function lines from the pipeline are operational.

5.1.8 REGISTER TEST

This routine will load a value into each register. The algorithm for this is to load a constant into the Q register, place q into register 2 (writing in reg 2) . Next take Register 2, add it to Q and place the result in register 3 (reading from 2, writing to 3). This is continued up to register F. The process is then reversed and the registers are now compared with their calculate values (calculated using the Q Register and Register 0).

5.1.9 FLAG TEST

This exercise will verify the ALU related 2-way branch functions. Among them are the Z8, Z4, C16, C8, C4, Y15, and Y0.
Each branch is exercised at both polarities. The inverted function of each branch need only be tested once since the hardware path of each case is the same (all go through the same XOR gate).

5.1.10 SHIFT TEST

This test will test the shifter (Q and RAM) connections in between the 4 2901's. Also the connections between Q (Qo, Q15, Ramo, Ram15) are tested. This is accomplished by sending alternate, adjacent and single bits patterns through the shifters. (alternate=$AAAA or $5555) (adjacent= $3333,$6666,..etc) (single= $1111,$2222,$4444..etc) Care is taken to make sure that Q doesn't shift during a single shift and that it does shift during a double shift. Polarity changes for the msb and the lsb of each nibble has also been accounted for. The patterns are modified somewhat in order to test each of the four shift modes.

5.1.11 STACK TEST

This first 4-way branch condition is tested here. This routine first starts out by popping the stack 16 times to insure emptyness. After that , 16 call's are made to insure fullness. At this time a simple call and return is made over a very confined address space (in case of error, hopefully it will be trapped in this space by the micro-code. After this is successful, 16 call's are made through the address space. At each return, a "return counter" is incremented in a manner similar to the jump test. Upon return to the main routine this counter is compared to the calculated number of returns to see if the stack had made the correct insertions on to the Microstore Address bus.

5.1.12 GPC TEST

This routine keeps in mind the physical construction of the GPC. It starts out by loading the GPC with a 0 and then 1 microcycle later it tests it for 0 (it takes 1 microcycle to latch the GPC). The next test is to test it as a 4 bit counter, then an 8 bit counter, etc. This will allow the user to trouble shoot down to the chip level since each counter in the GPC is 4 bits wide. An ALU register is also run in parallel with this to insure that the GPC does not bottom out too soon or too late.
This same technique is used for GPC2, the test is only for a 4 bit counter. This can only be tested by a 4-way branch.

5.1.13 D-SOURCE TEST

This is actually three tests in one since they are very short. The D-Source THREE is compared to a literal $3. The D-Source FFFF is compared to the literal $FFFF . In swap an alternating bit pattern $55AA in placed into SWAP , read back and checked, if correct, the read back value if placed back into SWAP and checked for the original value.

5.1.14 LOCAL CONTROL READBACK

Due to some of the effects that the Local Control Register has on some of the portions of the CPU, care is taken during the testing of each bit. For example, HALT is skipped completely since is is previously used during the Confidence Halts. Another this to consider is that the readback inputs to bits 2**2, 2**3, 2**5, 2**6, 2**7 are tied low and they are tested as such.
A shifting bit pattern is used to test the individuality of each line.

5.1.15 PROGRAMMABLE INTERVAL TIMER TEST

The technique used in this routine is similar to the GPC test since the physical construction of this counter is the same. One precaution must be made in that this clock in accordance to T0 and not T0H. The refresh HOLD memory control must be made in order to keep the ALU running in synchronous fashion to the PIT. Since the counter is readable, is is counted down along side a parallel register which periodically checks the value of the timer and compares it with itself.
After this functionallity is verified, HOLD is removed and PIT interrupt is allowed to check the interrupting circuitry.

5.1.16 BCD ADDER TEST

This routine basically tests the BCD ADDER rom more than anything else. The circuit must be tested in an add and subtract mode. The first thing that is checked is that the zone bit of the result is $3 . Next, the number 9 is added to 9, 9+8, 9+7.....8+9,8+8... 0+0). Carry from the previous operation is added to the present operation to check the carry logic. The subtract test is done in a similar manner. The alu also does the operation in a somewhat slower fashion to test the results from the adder. In that routine, the state of the carry bit must be read from the previous operation, added or subtracted in accordance to the mode and the state of the carry flags must be predicted by the alu to be checked after the operation

5.1.17 SPECIAL BRANCH TEST

The Special Branch test to check the ability of the Branch Register and the "Y" bus to place an address on the Micro-store Address Bus. This is done in a 16-way and a 256-way fashion. The bits are individually tested ie: a branch of +1, a branch of +2, a branch of +4, a branch of +8, etc...

5.1.18 GLOBAL AND MICRO-INTERRUPT TEST

Since some of the states of the elements in these 2 registers cannot be controlled, only the predictable elements will be tested. Start/Stop is the only bit that is tested on the Global "Register". Power Fail and Ipl-Warning are the only two bits that are tested in the micro interrupt "register"

5.1.19 BASE REGISTER TEST

This is the first time that the TEMP register is used. The BASE FILE is treated as memory and tested as such.
The patterns used are all $AAAA for odd numbered base registers and $5555 for the even numbered base registers. On each write the base pointer is reloaded thus bypassing the auto-increment on write function. On the shorted address line section, the base address is either written into the upper or lower section of the ram ie:

base reg	1	baseu= $0100	basel=$01
	2	baseu= $0002	basel=$02
	3	baseu= $0300	basel=$03
	etc	....	

5.1.20 LOGICAL ADDRESS TEST

This is the same for LARA,LARB, and LARP since the registers are read-writable ,$FFFF is loaded into a an ALU register and also to the LARX and then read back, this is decremented and repeated until 0 is reached.

5.1.21 REAL ADDRESS REGISTER TEST

This is more complicated for 2 reasons. 1) The register is 24 bits wide instead of 16 (alu width). 2) There is no direct readback path for this.
Problem 1 is solved by testing the real address register in two parts The upper 8 bits and then the lower 16 bits.
Problem 2 is taken care of through the D-Source PERR this will latch the address and save it until the next memory access. This is controlled by bits in the local control register.

5.1.22 PHYSICAL ADDRESS REGISTER TEST

The physical address test consists of three subsections. Each is identical except that each will test one of the three physical address registers (PARP, PARA, PARB).
To simplify circuit debugging, each subsection is divided into two parts. One part will check the upper 12 bits of the physical address (which is derived through the base adder adding $0000 to the upper 12 bits of the base value). The other part will exercise the lower 12 bits of the physical address register (which adds the base value to the logical address).
The second portion of the test loads the base registers with a shifting pairs pattern. The values are as follows:

5.1.23

BASE REGISTER	VALUE
0	000000
1	000001
2	000003
3	000006
4	00000C
5	000018
6	000030
7	000060
8	0000C0
9	000180
A	000300
B	000600
C	000C00
D	001800
E	003000
F	006000

This pattern will give the routine the ability to test for the independence of bits in the base adder and the physical address registers.

5.1.24 MEMORY TEST

This should find catastrophic errors in memory . It assumes that there is 64K in the machine and it does a stuck high and a stuck low bit test. It also will do a quick shorted address line test. All this is done through the REAL ADDRESS REGISTER in order to avoid using the mapping circuitry. After the memory is verified, the LOGICAL ADDRESS REGISTERS are used in order to access memory. This is done to test and verify the mapping circuitry.

6.0 FETCHER TEST

6.1.1 The Fetcher test takes up from where the quick memory test in the Q64 Diagnostic leaves off. It does make the assumption that the actual memory module is operating correctly. The approach that this diagnostic takes is to check that the fetch logic performs the correct functions.

6.1.2 The first section of the test will check that the memory contol can correctly do the various increment and decrement controls on the logical address registers. This is done without any alteration to memory.

6.1.3 After this the Cascade Registers are checked. (LCAS and RCAS). This is done by driving $F00F through the registers and checking and then by driving $0FF0 through the registers.

6.1.4 The D-Sources associated with the Fetch Register are checked by writing the pattern $0123456789ABCDEF out to memory and then reading it back in through the Fetch Register. After doing this the contents of OP1, OP2, OP3, CODE1, CODE2, RWORD2, RWORD3, RWORD4, and MOVLEN are checks for the correct values. If successful, the pattern is reversed (to check for bit errors) and the process is repeated.

6.1.5 The branch logic is checked next, the following sequence is loaded into memory:

PCTR	code	MNEMONICS
0000	0000A0	NOP $0000
0003	25A5A7	BRU $25A5
0006	5A5AA2	BMI $5A5A
0009	7FFFA3	BNZ $7FFF
000C	0000A4	BZ $0000
000F	3333A5	BNM $3333
0012	4CCCA6	BNO $4CCC
0015	5DDDA1	BOV $5DDD
0018	3BBBAA	BP $3BBB
001B	0000AF	BNP $0000

6.1.6 The P-counter is then set to 0 and a "Start the Fetcher" and a "Set Single Step Mode" is set to the Local Control Register. This will allow us to exercise the first branch and monitor the P-Counter. The Decode branch is then monitored checking for either of three conditions 1) Running (which will allow us to loop), 2) Complete and single step (which will exit the routine) 3) anything else is illegal and is considered an error at this point.

6.1.7 LARP and LARPO are checked after each operation for validity. If correct LARP and FLAGS are set to check the following condition. Each condition is checked in a "Branch Taken" and "Branch Not Taken" mode.

6.1.8 Next the indirect decrementer is tested. First in an overflow state and then in an non-overflow state. The first is accomplished by generating a branch instruction that will indirect to itself. The second part is accomplished by generating an indirect chain which is resolved at the fifteenth level (a non-overflow condition). Upon completion LARP is now checked to see if the resolved address is correct.

6.1.9 A Hazard on Instruction generated next by placing LARP within 8 bytes of the end of bank 0.

6.1.10 A similar method is used to generate Hazard on Indirect P and Hazard on Indirect B vectors.

6.1.11 It is assumed that at this point that the Fetcher can resolve an indirect address on "A" so these test cases are not generated for the single operand vector test. This routine in simple terms generates "CODE 1" values , calculates decode vectors, writes "CODE 1" into memory, sets the pcounter, starts the fetcher and finally checks the decode vector.

6.1.12 Double and triple operand instructions are generated and tested in a similar manner. The diagnostic generates a pattern of data that fulfills the requirements of the Fetcher (a counterfeit macro instruction). The macro instruction is loaded into memory, the p-counter is set to it and a fetch is initiated. Upon completion, the decode vector is first tested. After passing this LARP and LARPO are tested (LARP will increment either by 6 or 8). The resolving and identifying of indirects is also considered for each double and triple operand instruction.