EXTERNAL REFERENCE SPECITFICATION

PERIPHERALS

DIAGNOSTIC SYSTEM
(for disks)

Rev. 2.0

Darren Whittaker

30 June 1980

INTEL PROPRIETARY

Do Not Reproduce

TABLE OF CONTENTS

CHAPTER 1

PRODUCT INDENTIFICATION

1.1 NAME, MNEMONIC AND PROJECT NUMBER

Name: Peripherals Diagnostic System

Mnemonic: PDS

Project Number: 2908

1.2 PRODUCT ABSTRACT

PDS's purpose is to test Peripherals disk products such that a hardware problem can be found and reported. Also future expansion capablities for non-disk products are provided for, though not covered by this document.

1.3 PRODUCT USE ENVIRONMENT

1.3.1 Hardware

PDS operates in the MDs-800 and Series II (and the Series II-86 when PDS is converted to run on an 8086) with 64k bytes of Ram memory, a double density/single sided disk controller with 1 or more drives, line printer, and the drive subsystem under test (14in. Winchester an/or ustore controller each with one or more drives. One track on each surface will be devoted to diagnostics (referred to as the Diagnostic Track or Cylinder) and is not available to the users of these disk products. The Controller's Invocation Block Switches and Interrupt Switches must be set to the values given in Section 4.1.1.2 for PDS to be able to run the controller.

1.3.2 SOFTWARE

PDS is developed based on System Test Foundation Software (STFS) by John Paulson and A. Dain Samples which is loaded into memory and called for execution by ISIS-II. It relies on STFS for console output facitites and command line interpretation. PDS uses ISIS-II for its console input when the user is asked to answer the initialization questions.

CHAPTER 2

FUNCTIONAL SPECIFICATION

2.1 GENERAL CHARACTERISTICS AND SCOPE OF PRODUCT

PDS will be built around STFS. STFS provides an enviroment for product evaluation and for developing, debugging, and running system tests, diagnostic aids, and customer confidence tests. It also provides a standard human interface, as well as providing 'hooks' for easy implementation and debugging test and diagnostic routines.

The Diagnositc System can be divided into 3 types of commands: Test Manager, Procedure, and Utilities and Test routines and CALL Procedures. Appendix C illustrates in block diagram form the relationship of the commands, routines, and Procedures.

The Test Manager commands provide a mechanism for executing the test routines and reporting the results in a uniformed manner. The test routines are all non-destructive to the user data on the disks. The Test Manager by itself could provide the Customer Confidence Tests. Sections 4.2.1 and 4.3.1 describe the Test manager and the test routines.

The Procedure command is CALL. This command is used to execute one CALL procedure. The CALL Procedures provide a way to execute individual controller funtions, display read and write buffers, and to do some testing that can be destructive to the data on the disks. Sections 4.2.2 and 4.3.2 describe the CALL command and the CALL Procedures.

The Utility commands control the loading, starting, and exiting from the Disk Diagnostic System. They also provide a way to control the flow of command execution by providing the REPEAT, COUNT, and IF commands. One feature provided with the Utility commmands is the Macro capability. This feature permits the user to define a set of commands into one block and then using one command he can invoke the execution of all the commands in the block. Section 4.2.3 describes the Utility commands.

Three levels of error reporting are provided with all the test routines and most of the diagnostic routines. The first level is reported by just a PASS or FAILED message being displayed. The second level provides more information about the type of operation being performed by displaying an error code that helps to identify the controller function that caused the error. The error code is only displayed if a FAILED was returned and ERROR is set to a OFFH. The third level provides more status information by displaying certain types of status info depending on the type of error. This status is only displayed if a FAILED was returned and DEBUG is set to a OFFH. It should be noted that it is possible to have level two and three together, only one, or neither displayed. Appendix A contains the more detailed descriptions of the error messages. DEBUG and ERROR are described in Section 4.2.3.

2.2 DESCRIPTION OF MAJOR FUNCTIONS.

(1) PDS1

PDS1 countains the CALL procedures and is loaded into memory by the LOAD command so that its' symbols are also loaded.

(2) PDS2

PDS2 is the main module of PDS. It initializes the diagnostics by asking the user questions, controls which test is to be executed, resets the disk controller that is under test and sets up messages for display.

(3) PDS.OVx

These are the overlay files which contain the test routines to be executed (x is the overlay number) by the Test manager. See Section 4.2.1 for description of individual test routines.

2.3 PERFORMANCE CHARACTERISTICS

Actual performance of each individual routine/procedure depends on the function being performed and the type of drive under test. Each routine or procedure should be able to be run in under 3 minutes. Most will run in less than one minute.

2.4 APPLICABLE STANDARDS

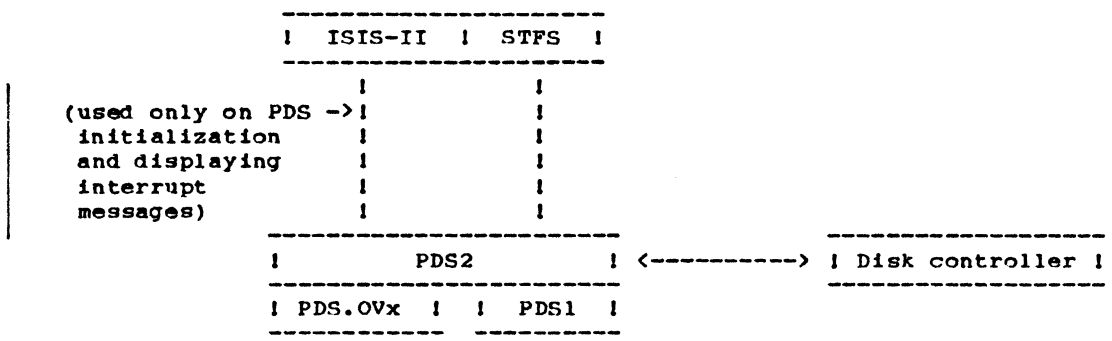
The command language used by PDS will conform to STFS) by D. Samples and J. Paulson (see Section 4.2 for commands used in PDS)

CHAPTER 3

INTERFACE SPECIFICATION

The disk controller's Hardware/Software interface will conform to "iSBC 215 Ustore Hard Disk Controller Functional Specification" by Don Goodrich. Refer to figure 1 for a block diagram of the interfaces.

FIGURE 1



3.1 INTERFACE TO STFS

The main diagnostic module, PDS2, interfaces with STFS. It uses the global variables and the output procedures that STFS provides. No detailed explaination of their usage is given here, for more information the reader is refered to the STFS ERS Sections 3.3.2 and 3.3.3

3.1.1 STFS's GLOABAL VARIBALES

STFS has a set of global variables that the user will need to know about. In some cases the variables will contain STFS information for the test routines, and in other cases the variables will be the means by which the test routines communicate with STFS. Below is a list of the variables, their PL/M declarations, a description of their use, and the range of legal values. For consistency all STFS variables are of type ADDRESS with the only exception of two components of the TDT structure. The DTFS variable ABORTED is the only byte variable public to the user.

STFSNOTCONF ADDRESS: this variable is TRUE (odd) if STFS is the program that is running, and is FALSE (0H) if CONF is running.

STFS$LAST$TEST ADDRESS: reports to STFS the number of the last test in the TDT; used by STFS both to insure TDT integrity, and to know the maximum size of the TDT; set by the user's main program when executed; the user is limited to 65536 tests, numbered 0 to 65535.

STFS$STATIC$TDT$POINTER ADDRESS: reports to STFS the address of the Test Descriptor Table; used by STFS to access the TDT; set by the user's main program; should never be changed; it must point to an address in the user memory space defined by UPPER and LOWER; the table should contain the following information for each test:

FLAG- a BYTE value used by STFS to keep track of various boolean values for this test (bit 0 = 0 if the test is RECOGNIZEd and 1 if it is IGNOREd; bits 1 - 7 are reserved by STFS);

OVERLAY- a BYTE value containing the overlay number plus one in which this test is found;

ADDR- an ADDRESS value containing the address of this test;

NAME$PTR- an ADDRESS pointer to a DTFS string that describes or names this test (all such strings are assumed to be in the root module loaded by INIT and not in overlays);

ERR$CNT- an ADDRESS value containing the number of times the test returned an error indication to STFS.

EXEC$CNT- an ADDRESS value containing the total number of times this test has been executed;

STFS$FOVERLAY ADDRESS: reports the overlay number plus one of the current FINISH routine (see section 3.2.4.4 in STFS's ERS); STFS will load (if necessary) and execute this routine when the user invokes the FINISH

command; can be set at any time by user programs; if there is no FINISH routine, STFS$FOVERLAY and STFS$ADDRESS both must be zero; this does not preclude the FINISH routine from being in the root, i.e. STFS$FOVERLAY = 0.

STFS$FADDRESS ADDRESS: contains the address of the current FINISH routine; forms a complementary pair with STFS$FOVERLAY; can be set at any time; if there is no FINISH routine, STFS$FOVERLAY and STFS$FADDRESS both must be zero; otherwise, the address must lie within user memory (i.e. between UPPER and LOWER).

STFS$TEST$BLOCK STRUCTURE: a public read-only table; through this table the user can access information about the invocation of this test run; this information includes:
TEST$NUM- an ADDRESS value containing the number of the current user test;
GO$TEST- an address value that is TRUE if the test was invoked via a GO FROM TEST, FALSE if the current test was invoked via the TEST command;
CNTONSFOR- an ADDRESS value that is zero if the TEST command had a COUNT clause, one if an ON clause, or two if a FOREVER clause;
CNT$ERR- an ADDRESS value that contains the value of the upper bound of the COUNT clause of the TEST command, or it contains a zero if the ON clause was NOERROR, and a one if the ON clause was ERROR;
CUR$CNT- an ADDRESS value containing the current iteration number of the TEST COUNT command.
NOF$PASSES - an ADDRESS value containing the number of times the TDT has been scanned since the last CLEAR command; whenever a TEST command is invoked, STFS makes note of which tests are to be run and then scans the TDT for a test to run; NOF$PASSES records the number of times the scan was restarted.

STFS$V(16) ADDRESS: this array is for communication between STFS, the user's program, and the console; corresponds to the predefined user variable V defined in section 3.2.3.5 (in STFS's ERS); they can be set at any time; their value and interpretation is left to the discretion of the user.

STFS$DEBUG ADDRESS: corresponds to the predefined user variable DEBUG; can be set from the console or from a user program; checked by STFS to determine whether to print a user generated output; can have the value zero (RESET, no debug), or 0FFFFH (SET, debug output printed).

STFS$ERRORONLY ADDRESS: corresponds to the predefined user variable ERRORONLY; can be set from the console or from a user program; checked by STFS to determine whether to print a user generated output; can have the value zero (RESET, print all messages but debug messages) or 0FFFFH (SET, print only error messages).

ABORTED BYTE: a DTFS global variable that is set when an escape key is typed from the console; this is a read-only variable.

3.1.2 STFS PROCEDURES

STFS will provide various useful functions, particularly for I/O. Below is a list of the routines available , their PL/M declaration, description of their parameters, and results to be expected.

STFS$CHANGE$RST: PROCEDURE (VAL); instructs STFS to change the restart instruction used for breakpoints to the one that will use interrupt VAL, an address value that must evaluate to be less than eight; if VAL is an illegal value, the routine will abort and return control to STFS; using this routine anywhere but in the main module of a file being INITed will cause unpredictable execution.

STFS$START: PROCEDURE; returns control of the system to STFS; the current command is aborted (with an attendant message), and the user is prompted for the next command; required in the user's main module; breakpoints, if in use, will be reset.

STFS$ABORT$CHECK: PROCEDURE; a routine that checks for the escape key; if the escape key has been typed by the user from the console, the current test is aborted as gracefully as possible (e.g. breakpoints, if in use, will be reset) and control is returned to STFS; the user's code should call this routine occasionally to prevent a long delay between the time the escape key is hit and control returns to STFS; however, if the test must not be interrupted, this routine should not be called by the user: it will be checked between tests automatically.

If the user must do some clean-up himself before returning control to STFS he may check the DTFS boolean variable ABORTED directly. This variable is set TRUE when an escape key is detected. The user can then perform the necessary clean-up, and return control to STFS via the STFS$START procedure.

STFSCANPRINT: PROCEDURE(MSG$TYPE) BOOLEAN; where MSG$TYPE is a byte as described below. Returns TRUE (odd) if a message of type MSG$TYPE may be displayed, FALSE otherwise.

STFS$DISPLAY: PROCEDURE(PTR,MSG$TYPE); where PTR is the address of a DTFS string, and MSG$TYPE is a BYTE variable denoting the level of the message: if the message is a status message, bit 0 is set; if an error message, bit 1 is set; if a debug message, bit 2 is set. If user variable ERRORONLY is TRUE (i.e. the low order bit is set), and bit 1 of MSG$TYPE is set, then the message will be printed. Else, if user variable ERRORONLY is not TRUE, and user variable DEBUG is TRUE, and bit 2 of MSG$TYPE is set, then the message will be printed. Else, if bit 0 of MSG$TYPE is set, then the message will be printed. Otherwise, the message will not be printed. The

following table summarizes the action of STFS$DISPLAY:

			ERRORONLY			
			false		true	
MSG$TYPE			DEBUG			
debug	error	status	false	true	false	true
0	0	0	NO	NO	NO	NO
0	0	1	YES	NO	NO	NO
0	1	0	NO	NO	YES	YES
0	1	1	YES	NO	YES	YES
1	0	0	NO	YES	NO	NO
1	0	1	YES	YES	NO	NO
1	1	0	NO	YES	YES	YES
1	1	1	YES	YES	YES	YES

An equivalent statement would be:

```
IF (ERRORONLY AND BIT1) OR
   (NOT ERRORONLY AND DEBUG AND BIT2) OR
   (NOT ERRORONLY AND NOT DEBUG AND BIT0) THEN PRINT;
```

Note that STFS$DISPLAY is equivalent to DISPLAY (described below) with additional checking to determine whether to print the string.

The following routines are from DTFS and do not check the STFS variables DEBUG and ERRORONLY. These routines output the specified information immediately to the console and to the listing file, if in use. These routines keep track of the position of the cursor, allowing tabbing and some screen formatting.

DISPLAY$CHAR: PROCEDURE(CHAR); where CHAR is a one-byte character.

NEW$LINE: PROCEDURE; if the output is not currently at the beginning of a line, a carriage return/line feed is output.

DISPLAY$NUMBER: PROCEDURE(VALUE,FORMAT); where VALUE is a sixteen bit value and FORMAT a byte specifying how the number is to be printed according to the following encoding:

```
bits 7,6 = 00 binary     bit 5 = 1 - append suffix
           01 octal            4 = 1 - suppress leading zeros
           10 decimal        3,0 = number of digits-1 to print
           11 Hexadecimal
```

DISPLAY: PROCEDURE(TEXT$ADDR); where TEXT$ADDR is the address of a DTFS string (see section 3.3.1 in STFS's ERS).

TAB: PROCEDURE(C); where C is a byte value specifying the column to which spaces are to be printed.

DISPLAYINBASE: PROCEDURE(NUMBER,SIZE); where NUMBER is a sixteen bit value and SIZE a byte value specifying whether NUMBER is to be treated as a bit value (SIZE=0), a byte value (SIZE=1), or a word value (SIZE=2).

```
DISPLAYSSPACE: PROCEDURE; prints one space.
```

3.2 INTERFACE TO ISIS-II

PDS uses two ISIS-II system procedures, READ and CO.

The READ procedure is used to input, from the console, the user's answers for the initialization questions asked by PDS.

```
READ:  Procedure  (AFTN,Buffer,Count,Actual,Status);

       AFTN (active file or device number) will equal 1 for :CO:.
       Buffer will contain the address of the buffer that will contain
         the data read.
       Count is the number of characters expected to be input.
       Actual is the actual number of characters inputed.
       Status is an address of a memory location in which ISIS will
         store any non-fatal error number.
```

The CO procedure is used to display interrupt messages to the console. The two messages displayed are for a seek complete and pack change and will only be displayed if PDS variable V(1) equals 0FFH.

```
CO:  Procedure (Char);

       Char is the ascii character that is to be displayed on
         the console.
```

The reader is refered to the "ISIS-II USER'S GUIDE", page 5-10 for more information on the READ procedure and page 5-32 for the CO procedure.

3.3 INTERFACE TO THE DISK CONTROLLER

All disk operations are initiated by the host system's Central Processor Unit (CPU) communicating with the controller via a controller invocation block in system memory. Once initialted, the disk controller completes the specified operation without further intervention on the part of the CPU and when the operation is complete, the CPU is notified via the same controller invocation block and/or by an interrupt posted by the controller.

For a more detailed description the reader is referred to "iSBC 215 uSTORE Hard Disk Controller Functional Specification" by Don Goodrich, Section 4.2

3.4 USER INTERFACE

The user interfaces to PDS by entering commands to the console device. The system indicates that it is expecting a command by prompting with an asterisk ("*") on the console. The user then enters a sequence of characters, which are grouped into tokens according to lexical rules, which in turn form commands according to the syntax given in Section 4.2.

PDS commands consist of one or more command lines, each terminated by at least one of two line terminators: carriage return or line feed. A command line may contain intermediate line terminators (i.e., a command line may extend over more than one input line) as long as the line terminator is contained in a string, or an ampersand ("&") not contained in a string or a comment precedes each intermediate line terminator. Characters between the ampersand and line terminator are ignored, and the ampersand is treated as a space. The system indicates that it is ready to accept a continued command line by prompting with two asterisks ("**"). The prompts issued within compound commands are described in section III.E of the Diagnostic Tool Foundation Software (DTFS) ERS. Each input line may contain no more than 120 characters before the line terminator (an ISIS restriction).

PDS uses the ISIS-II line editing capabilities to correct errors in an input line. The line-editing characters are:

Rubout	delete last character type in input line.
Control-x	delete entire input line.
Control-R	echo entire input line.
Escape	cancel entire input line.
Control-P	input next character literally.
Control-Z	delete entire input line.
Carriage-return	end of input line.
Line-feed	end of input line.

Once a line terminator has been entered, that line can no longer be edited.

For more information on the user interface the reader is referred to STFS ERS Section 3 (Lexical elements are defined in Section 3.2.1).

CHAPTER 4

OPERATING SPECIFICATIONS

4.1 PRODUCT ACTIVATION INSTRUCTIONS

4.1.1 ENVIRONMENT NEEDED

As specified in 1.3.1, with the Controller Invocation Block switches set to 9000H and the interrupt switches set to cause a Series II interrupt 5 when the controller causes an interrupt.

4.1.2 ACTIVATION SEQUENCE

All numbers entered to answer any of the following question must be in decimal.

1. Boot system off of double density disk drive under ISIS-II
2. Enter STFS, wait for * (STFS's prompt)
3. Enter LOAD PDS1, wait for *
4. Enter INIT PDS2, PDS will now ask the following questions with the user responding accordingly.

    a. Enter a hex random number seed between 0 and 64k.
    b. Type of unit being tested ? Flopp(0), 8in(1), or 14in(2) - Enter Number.
    c. Is unit x (x = 0,1,2,3) being tested (Y or N)
                (If yes the next three questions are asked.)
       Is this unit backed-up(Y or N)
       Does this unit have 2 volumes(Y or N)
                (Only asked for 14in units.)
       Do you want to use the initialization defaults(Y or N)
                (If no the next 6 questions are asked.)
          Number of Tracks/Surface ?
          Number of Fixed Surfaces ?
                (Not asked when testing floppies.)
          Number of Removable Surfaces ?
                (Not asked when testing 8in. units.)
          Number of Sectors/Track ?
          Number of Bytes/Sector ?
          Number of Alternate Tracks ?
                (Not asked when testing floppies.)

    C. is asked once for each of the four possible drives that can be attached to the controller under test. If a carriage return is entered as the answer to any of the question (except b.) a No or a default number (which is displayed) is assumed.

    If a 'B' is entered for any of the above questions the previous

question will be asked again.
5. Enter the command of your choice (see section 4.2 for PDS commands).

4.2 COMMANDS

****

Since PDS is based on STFS all of STFS's commands can be used. The commands listed here are the only STFS commands the user will need to use PDS.

****

4.2.1 TEST MANAGER COMMANDS

```
Test_comnd = "TEST" [ Test_ranges ]  [ Test_condition ] .

Test_condition = "COUNT" Arith_exp
     ¶  "ON"  Test_error_condition
     ¶  "FOREVER"
     .

Test_error_condition =  "ERROR"  ¶  "NOERROR"  .

Test_ranges = Partition "," ... .
```

The TEST command loads and executes a sequence of software procedures (i.e. tests routines) by number. The Test_ranges specified must evaluate to less than the total number of tests. If a number is specified for which no test exists, an error results. The tests are executed in numeric order regardless of the order in which they were specified in the TEST command.

If the COUNT phrase is included, the tests specified in the list of test ranges are executed in numeric order Arith_exp number of times. If Arith_exp is zero, the tests will not be executed, although the first test that would have been executed will have been loaded into memory.

If FOREVER is specified, the tests specified in the test range list will be executed in numeric order until the user hits the escape key. Unless the user's tests occasionally call STFS$ABORT$CHECK (or check the global variable ABORTED themselves; see ???), striking the escape key will only take effect after the current test finishes and before STFS initiates the next test.

If ON ERROR is specified the range of tests will only be repeated if at least one of the tests returned an error condition to STFS.

If ON NOERROR was specified, the range of tests will not be completed if a test returns an error condition to STFS.

If a Test condition is missing, the default is COUNT 1.

Summary_comnd = "SUMMARY" [Test_ranges] ["EO"] .

The SUMMARY command is conceptually a part of the Test Manager portion of STFS. When a test group has been run via the TEST command PDS will keep a record of the history of the run. For each test known to PDS within the specified range the following information will be displayed by the SUMMARY command: the test number, the name of the test, the number of times the test was executed, the number of times this test returned with an error indication, and whether the test was ignored or not. If Test_ranges is missing, all tests will be included in the SUMMARY display. If EO (erroronly) is included, then only those tests with a non-zero error count will be displayed.

The summary listing will be concluded with a statement as to whether any of the tests show a non-zero error count, and the total number of passes.

Clear_comnd = "CLEAR" [Test_ranges] .

The CLEAR command resets the counter values in the user's Test Descriptor Table, and the number of passes counter in the Test Parameter Block. For each test specified, or for all tests if Test_ranges is missing, the execution count and the error count are reset to zero. CLEAR does not affect the status (ignored or recognized) of a test, nor does the status of a test affect the CLEAR command.

Describe_comnd = "DESCRIBE" [Test_ranges] .

The DESCRIBE command will display the user defined name, or description, of the specified tests, and whether the test would be ignored by the TEST command. If Test_ranges is missing, the descriptions of all tests will be displayed.

Ignore_comnd = "IGNORE" Test_ranges .

Recognize_comnd = "RECOGNIZE" Test_ranges .

PDS provides the ability to tell the Test Manager to ignore and not run tests even if the tests are in a test partition in the TEST command. This allows the user at the beginning of an PDS session to declare via the IGNORE command which tests are not to be run under any circumstances. The RECOGNIZE command provides the inverse function of the IGNORE command.

4.2.2 PROCEDURE COMMAND

```
call_comnd = "CALL"  .diagnostic routine name  [ Parameters ] .

Parameters = "("  Exp  [ ","  Exp ] ")" .
```

The CALL command executes the diagnostic routine (via a PL/M indirect call). One or two parameters may be passed using the PL/M conventions, and are taken to be ADDRESS type variables. Return of the procedure returns control to the console.

4.2.3 UTILITY COMMANDS

Init-comnd = "INIT" Pathname.

The INIT command loads and runs the PDS2 initialization code and prepares PDS for running by asking the user questions concerning the type and number of drive(s) to be tested. Pathname must be PDS2.

Load_comnd = "LOAD" Pathname .

The LOAD command fetches the local symbols, source statement numbers, module names, and object code in ISIS-II file Pathname. Pathname must be PDS1.

List_comnd = "LIST" Pathname .

The LIST command causes a copy of all output, including prompts, input line echo and error messages, to be sent to the ISIS-II file Pathname. (If the specified pathname is :CO:, then there is effectively no list file, which is the initial setting.)

Exit_comnd = "EXIT" .

The EXIT command causes the PDS session to end and returns control to ISIS-II.

Error-comnd = ERROR ["=" (0 or 0FFH)]

ERROR without any parameter will display its' current value. ERROR with a parameter will set ERROR to that value. A value of zero will turn off the displaying of error codes. These codes are displayed if an error occurs while running a test or diagnostic routine. A value of 0FFH will turn on the displaying of the error codes and will stop the displaying of PASS data. The boot-up default is zero. For a description of the error codes see Appendix A.

Debug-conmnd = "DEBUG" ["=" (0 oror 0FFH)]

DEBUG without any parameter will display it's current value. DEBUG with a parameter will set DEBUG to the new value. A value of zero will turn off the displaying of status information. This information is displayed if an error occured while a test or diagnostic routine was running. This status helps to provide more information than is provided by the error code. A value of 0FFH will turn on the displaying of the status information. The boot-up default is zero. For a description of the types o status information provided see Appendix A.

Interrupt-message-flag = "V(1)" ["=" (0 or 0FFH)]

V(1) is a PDS variable that when set to OFFH will display a message when an interrupt occurs if the interrupt was caused by either a pack change or seek complete.

Compound Commands - A compound command is a control structure that contains zero or more commands. For a detailed descriptions of the following commands the reader is referred to the "ICE-86 In-Circuit Emulatore Operating Instructions for ISIS-II Users" manual pages 8-1 to 8-8.

REPEAT This command executes zero or more commands in a loop. WHILE and UNTIL are keywords used to control terination of the loop.

COUNT COUNT is like REPEAT though it has a loop counter that will terminate the loop if no exit condition (using WHILE or UNTIL) is met before the counter runs out.

IF The IF commands permits conditional execution in a command sequence.

Macro Commands - A macro is a block of commands. When a block of commands is defined as a macro, it is stored on diskete so that it can be executed more that once without having to enter the commands each time. For a detailed description of the following command the reader if referred to "ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users" manual, pages 8-8 to 8-17.

DEF MAC This command defines a macro. Once defined the macro can be invoked (called) by placing a ":" in front of the macro's name.

MAC MAC will display the macro's definition.

DIR MAC This command will list the names of all the macros that have been defined.

REM MAC This command will delete a macro.

PUT MAC This command causes one or more macro definitions to be saved in a disk file for use in the next diagnostic session.

INCLUDE INCLUDE loads the macros that were saved on a file by the PUT MAC command back into memory.

4.3 ROUTINES and PROCEDURES

4.3.1 TEST ROUTINE DESCRIPTIONS

TEST 0 RESET DISK TEST
This test will reset the disk controller by sending the controller a reset and a channel attention signal. This will place the device in a known state. This test should return a Pass value in under 2 seconds.

TEST 1 TRANSFER STATUS TEST
This test checks the basic handshaking of the controller and makes sure that the units attached can by selected by executing its transfer error status function for each unit attached.

TEST 2 BUFFER I/O TEST
This test will verify that the transfer of data to and from the controller is working correctly.

TEST 3 CHECKSUM TEST
This test calculates the checksum of the firmware by summing with carry added in the contents of all Rom locations (excluding the stored checksum). It then compares the calculated checksum value to the original checksum stored in the Rom. Each rom has its own byte checksum so you will have a checksum for the even bytes and another checksum for the odd bytes.

TEST 4 RAM WINDOW TEST
All of Ram is set to zero with only one word set to all ones (0FFFFH). This value will be walked through all of memory. After every move all of Ram is checked to see if this word of ones at that location has any diverse effect on any other location. Then all of memory is set to 0FFFFH with only one location being set to zero. This value is then walked through memory and all of memory is checked after each move.

TEST 5 RAM ADDRESS TEST
This test will copy the contents of Rom into Ram and then compare them. Then the contents of Ram are inverted and ANDed to the contents of the Rom which should produce a value of zero. This test will ensure that the Ram address lines are working and that each bit of Ram can be turned on or off.

TEST 6 MICRO-DIAGNOSTIC TEST
This test runs the on board controller micro-diagnostics. This test recalibrates and then seeks to the last cylinder. After doing a read ID to ensure that the heads are over the last cylinder a write to sector 0, head 0 is performed followed by a read verfiy. This is done for each unit attached.

TEST 7 SEEK/VERIFY TEST
This test verifies sector zero on the last cylinder, head 0 and then on cylinder 0, head 0. This is done for every unit attached.

TEST 8 FORMAT DIAGNOSTIC TRACK
This test will format the diagnostic track with an interleave factor of 4 and the sector size determined at PDS initialization.

TEST 9 WRITE/READ DIAGNOSTIC TRACK TEST
This test transfers a predetermined number of sectors of contiguous data from system memory to the diagnostic track. After the track is written a read is performed and the write and read buffers are compared. This test is performed on all available drives, platters and heads. Note that the diagnostic track must be formatted before running this test.

TEST A DRIVE SELECTION TEST
This test will verify that each drive can be addressed correctly. All drives which are available for testing (as indicated in the invocation) will be checked to see that they are ready. The diagnostic track (designated platter, head 0) on each of these drives will be formatted with different data after which the formatted data will be read and compared with data written. All errors will be reported.

TEST B PLATTER/HEAD SELECTION TEST
This test will verify that each platter and head can be addressed uniquely. The diagnostic track of all available drives (all platters and heads) will be formatted with unique data. The formatted data will be compared against data written. Any errors will be reported.

TEST C SECTOR SELECTION TEST
This test will verify that each sector of a track can be addressed uniquely. Testing will be performed on the diagnostic track (head 0, designated platter) of all available drives. The sector number will be written into each sector. Each sector will be read and compared with the sector number.

TEST D TRACK VERIFY TEST
This test verifies a predetermined number of sectors of data from specific tracks. The platters verified must be formatted. This is a non destructive test.

TEST E OVERLAP SEEK TEST
This test verifies that the controller can properly handle overlapped operations, (excluded if only one drive is present). This test will operate on the diagnostic track of two of the available drives, head 0. The two drives are recalibrated. One drive executes a seek to the diagnostic track and then the other drive will do a verify of a sector on track 0.

TEST F WRITE AND READ DELETED DATA (FLOPPY DRIVES ONLY)

This test verifies that the write and read deleted data function of the floppy drives are working properly. This test will write deleted data to the diagnostic track and then read the deleted dat and compare what was written to what was read. All errors will be reported if DEBUG is set. This routine will return a FAILED if the floppy drives are not being tested.

4.3.2 CALL PROCEDURE DESCRIPTIONS

The CALL Procedures are divided into four categories, 1) Select Procedures, Controller Function Procedures, Test Procedures, and Misc. Procedures.

The Select Procedures allow the user to pre-define certain variables that are used by the other types of CALL Procedures. STFS ony allows two parameters to be passed th the CALL Proceduress. When a CALL Procedure needs more than two parameters it will use one (or more) of the values that have been defined by one of the Select Procedures.

The Controller Function Procedures each do one controller function. This provides the user to access the controller on a function by function basis.

The Test Procedures are procedures that perform some type of useful testing to the controller and the disks. They should have been included as TEST under the Test Manager but they are destructive to the user data.

The Misc. Procedures provide ways to look at and modify system memory, display the controller's I/O Parameter Block, and compare the user select read and write buffers.

4.3.2.1 SELECT PROCEDURES

SEC$CT (Count)
  Select the sector count.

SEL$CYL (Cylinder)
  Select the default Cylinder.

SEL$HD (Head)
  Select the default Head.

SEL$INTR (Interrupt$Retry)
  Select the default Interrupt and Retry modes.

SELRDBUF (Address)
  Select the beginning Address of the user selected Read buffer.

SELWRTBUF (Address)
  Select the beginning Address of the user selected Write buffer.

SEL$UNIT (Unit)
  Select the default unit.

4.3.2.2 CONTROLLER FUNCTION PROCEDURES

CTOS (Address,Byte$count)
  Write from controller memory to user selected read buffer the number of bytes specified by Byte$count.

FMTRK (Typeofformat,Interleave$factor)

Format preselected track. Uses bytes 0-3 of user selected write buffer for sector data field.

ECC (Address,Count)
This routine calculates the ECC remainder for the number of bytes specified by Count starting at Address.

RDID (Unit)
Read and display track ID on selected unit from the last track accessed.

ISEEK (Cylinder,Head)
Initiate a seek to Cylinder on preselected unit. Cylinder and Head become the new default values.

RSET
Reset the controller by causing a reset, followed by an interrupt clear, followed by a channel attention signal to the controller.

RD (Cylinder,Beginning$sector)
Read sector(s) from preselected unit and track into user selected Read buffer. The number of sector read is set by SEC$CT.

DELRD (Cylinder,Beginning$sector)
Same as RD except the data read is deleted data. (Floppy only)

SEEK (Cylinder,Head)
Seek to Cylinder on preselected unit. Cylinder and Head become the new default values.

STAT
Transfer Error Status and Sector address information from the controller and display it if any bits are set in the Status.

STOC (Address,Byte$count)
Write from user selected write buffer to controller ram to the location specified by Address the number of bytes specified by Byte$count.

VRF (Cylinder,Beginning$sector)
Verify sector(s) on selected unit and track. The number of sector is set by SEC$CT.

WSEEK (Unit)
Wait for the seek to complete on Unit. Usually used after ISEEK.

WRT (Cylinder,Beginning$sector)
Write data to sector(s) on preselected unit and track from user selected write buffer. The number of sectors is set

by SECtCT.

DELWRT (Cylinder,BeginningtSector)
Same as WRT only data written out is marked deleted. This is a function only for the Floppy.

4.3.2.3 TEST PROCEDURES

ALTSEEK (Cylinder1,Cylinder2)
Causes a continues seeking to occur between Cylinder1 and Cylinder2. The user must enter an escape to stop this procedure.

FMPLT (TypetoftFormat,InterleavetFactor)
Uses bytes 0-3 of the user selected write buffer for the data field in formatting the selected platter.

FMDRV (TypetoftFormat,InterleavetFactor)
Uses bytes 0-3 of the user selected write buffer for the data field in formatting the selected unit.

DKSK
This routine exercises the seek and verify funtion by causing the heads to move using the follow algorithm:

```
Do Forever;
     N = first track;
     M = last track;
     Do until N = last track;
          verify N;
          verify M;
          N = N + 1;
          M = M - 1;
     End;
End;
```

RANDOM (number-of-sectors)
This routine will randomly write to different sector(s) (the actual count based on the selected value) on the selected platter. It will write the sector, head and track ID and the rest of the data field will be filled with A's. Then using the same random number sequence the same sector(s) will be read and compared to the data written.

RECAL
Causes a recalibration of the selected unit to cylinder 0. Cylinder 0 becomes the new default cylinder.

RNSK
Will randomly seek on the selected unit until the user enters an escape.

TRKCK (Sector)
This routine will verify that all track address bits can be addressed correctly. For the selected drive (designated platter, head 0) all of tracks will have their Sector written to. The track number will be written into every byte of the sector data field. When the writing is complete the data on each track will be compared with the track number.

WRC (Cylinder,Beginning$sector$number)
Read after write then compare on pre-selected drive, platter, and head.

4.3.2.4 MISCELLANEOUS PROCEDURES

CMP (Count)
The user selected read and write buffers are compared on a byte by byte basis for the number of bytes specified by Count. If DEBUG = OFFH any mis-compares are will be displayed.

DIS (Address,Count)
System memeory, starting at Address and going for the number of bytes specified by Count, will be displayed.

DSIOPB
Displays the I/O Parameter Block. (See 'iSBC215 uSTORE Hard Disk Controller Functional Specification' by Don Goodrich for the definition of the IOPB.

FLRD (Word$count,Word$of$data)
The Wordofdata is repeatedly copied into succeeding locations of the user selected read buffer for the number of times specified by Word$count.

FLWRT (Word$count,Word$of$data)
Same as FLRD except the Wordofdata is repeatedly copied into the user selected Write buffer.

PAUSE (1/10$millisecond$count)
Just waits the number of counts speicified. Each count is one tenth of a millisecond.

APPENDIX A

ERROR MESSAGE DESCRIPTIONS

A.1 ERROR CODES

The following error codes are displayed if an error occured (FAILED) and ERROR is set to 0FFH. The code can help to pin point what function the error occur in.

CODE	DESCRIPTION
FORMAT	Format error.
LBUF	I/O buffer error.
DIAG	The micro-diagnostics returned an error.
READID	Read sector ID error.
READ	Read error.
RESET	Reset error.
SEEK	Seek error.
TRANST	Transfer Error Status error
WRTBUF	Write controller buffer to disk error.
WRITE	Write error.
VERIFY	Verify error.

A.2 STATUS MESSAGES

The following status messages will be displayed if DEBUG equals OFFH and the error bit is set in the Controller Invocation Block (CIB) Opeational Status Word (see "iSBC 215 Ustore Hard Disk Controller Functional Specification", by Don Goodrich.

u UNIT NUMBER
eh bbbbbb CIB STATUS (in binary, e is error bit and h is hard error bit)
bbbbbbbbbbbbbbbb HARD ERROR STATUS ** (in binary, will always be displayed plus zero or more of the following hard error messages).

HARD MESSAGE	DESCRIPTION
DIAGNOSTIC FAULT	A mircro diagnostic fault has been detected.
INVALID COMMAND	The controller has been issued an invalid function or other parameter which would cause an illegal function.
INVALID ADDR	An attempt to access a cylinder beyond the available tracks (including alternates) was made.
NO INDEX	The controller can not find the index mark for this unit.
RAM ERROR	The Ram is not functioning correctly.
ROM ERROR	Rom checksum error.
SEEK IN PROGRESS ERROR	The unit can not perform the requested function becaue it is already doing a seek.
SELECTED UNIT NOT READY	The selected unit is either not ready, not connected, or not responding to a unit connect.
WRITE PROTECTION FAULT	An attempt has been made to write to a write protected unit.
END OF MEDIA	End of media has been detected.
ILLEGAL FORMAT	The alternate cylinder can not be mark defective too.
SECTOR NOT FOUND	The desired sector was not found.
ILLEGAL SECTOR SIZE	The initialized sector size does not agree with the formated sector size.

bbbbbbbb SOFT ERROR STATUS ** (in binary, will also be displayed plus zero or more of the following soft error messages.)

SOFT MESSAGE	DESCRIPTION

DRIVE FAULT	A hardware fault has been detected in the selected drive and is characterized by: Read/Write fault, position fault, power fault, or speed fault.
DATA FIELD	A correctable error was detected in the data field of a sector - note retry count.
ID FIELD	A correctable error was detected in the ID field of a sector - note retry count.
CYLIN. ADDR MISC	Cylinder address miscompare - the ID field read contains a cylinder address different from the expected cylinder address.
SEEK ERROR	A seek error has been detected.

Then the following track addressing data is displayed.

	Desired	Actual
cyl	xxxx	xxxx
head	xx	xx
Sect	xx	xx

Number of retries - xx

A.3 TIME-OUT ERRORS

The following message are given if the controller does not respone within the amout of time it should.

NON-INTERRUPT TIME-OUT	The controller has not given an operation complete for a function that had inhibited the operation complete interrupt.
INTERRUPT TIME-OUT	The controller has not interrupted for an operation complete.
TIME-OUT ON SEEK COMPLETE	The controller has not posted a seek complete.

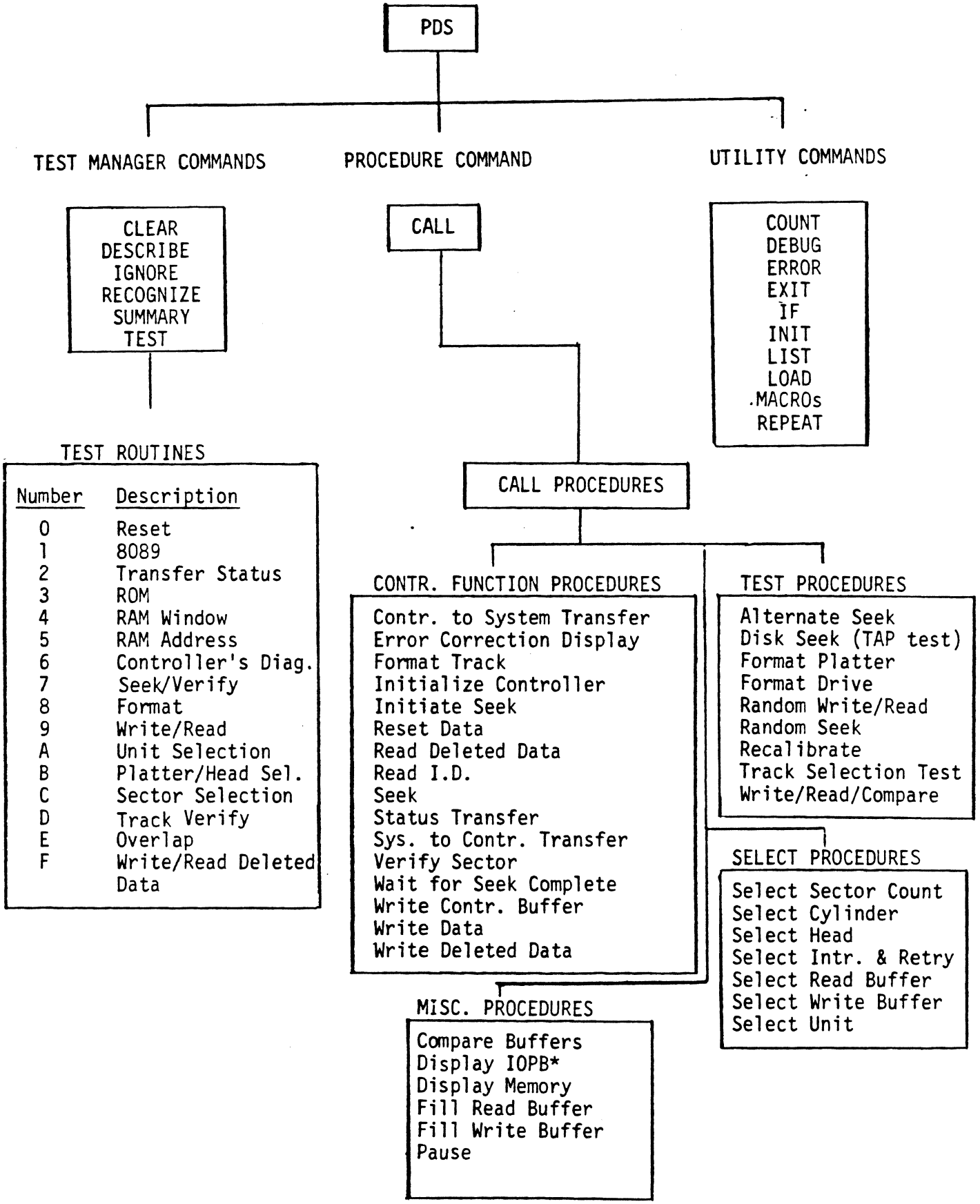
A.4 BUSY ERROR

The requested function could not be initiated because the controller's channel one is already busy and it should not be.

APPENDIX B

DRIVE DEFAULT VALUES

TBD.....



*IOPB - I/O Parameter Block