$1.50

SDS 940
OLDS DIAGNOSTIC SYSTEM

REFERENCE MANUAL

SDS 901591-A00 February 1969

SDS

SCIENTIFIC DATA SYSTEMS • 701 South Aviation Boulevard • El Segundo, Calif., 90245 • 213/772-4511

RELATED PUBLICATIONS

LISTINGS

OLDS 3.0 Control Monitor	870029-51A
UNIT 0 CPU 0 Diagnostic	870030-51A
UNIT 1 CPU 1 Exerciser	870031-51A
UNIT 2 FPAU Diagnostic and Exerciser	870032-51A
UNIT 3 MEM 1 Diagnostic for the 2nd 16K	870033-51A
UNIT 4 MEM 2 Diagnostic for the 3rd 16K	870034-51A
UNIT 5 MEM 3 Diagnostic for the 4th 16K	870035-51A
UNIT 12 RAD Diagnostic for Channel E	870036-51A
UNIT 15 RAD Diagnostic for Channel W	870037-51A
UNIT 21 DISC Diagnostic for Channel W	870038-51A
UNIT 23 CTE 10/11 Diagnostic	870039-51A
Updating Instructions	870029-61A

1. PROGRAMMING

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

1.1 OFF LINE DIAGNOSTIC SYSTEM CONTROL PROGRAM (OLDS)

This document outlines the 940 Off Line Diagnostic System Control Program. A brief summary of its objectives and structure will be given. However, particular emphasis will be placed on the OLDS syntax, the interface requirements between OLDS and its component 940 diagnostic programs and the operating procedure for OLDS.

For the remainder of this document the word "control" refers to the System Control program and the word "operator" refers to the person running or using the Program.

1.1.1 OBJECTIVE

The purpose of OLDS is to design a flexible control feature in a compact program, providing a common interface between the operator and all 940 diagnostic programs. This common interface has the following advantages:

Simple syntax, common to all 940 diagnostic programs.

Standard operating procedure, common to all 940 diagnostic programs.

Uniform breakpoint settings, common to all 940 diagnostic programs.

Common interrupt, trap and POP receivers.

Common input/output handlers.

These advantages will enable an operator to run any 940 diagnostic program by merely knowing how to communicate with OLDS.

1.1.2 STRUCTURE

OLDS is designed to handle four logical program control levels. These are defined to be:

System level program

Unit level program

Function level program

Object level program

These levels are designed to enable program modularity and ease of modification. The definition associated with each level is as follows:

System level program: The set of all 940 diagnostic programs, OLDS, and any 940 configuration.

Unit level program: A program which will completely diagnose, exercise and adjust an entire unit of a 940 system. A unit is a major subset of the System. A unit is always confined to 16K of core.

Function level program: A self-initializing program designed to diagnose, exercise and/or adjust a particular portion of a device of a 940 system. A Function is a major subset of a Unit.

Object level program: An autonomous self-initializing body of code, devoted to diagnosing a module, a small portion of back panel wiring or a cable.

An Object program is a major subset of a Function program and the smallest denomination of the System.

1.1.3 CONTROL LANGUAGE (SYNTAX)

1.1.3.1 Level Syntax

The following characters specify the four control levels:

S	System level control
U	Unit level control
F	Function level control
O	Object level control

1.1.3.2 Abstracts

A short explanation of each level can be requested by calling for the abstract.

SA	Calls for the System abstract
UA	Calls for the Unit abstract
FA	Calls for the Function abstract
OA	Calls for the Object abstract

1.1.3.3 Identifiers

A short title concerning the level is printed as the identifier.

SI	Calls for the System identifier message
UI	Calls for the Unit identifier message

FI	Calls for the Function identifier message
OI	Calls for the Object identifier message which is the starting core address for the particular object test.

1.1.3.4 Lists

A list of all identifiers is output.

SL	Not allowed. An error flag is printed. (✓ check mark)
UL	Unit list – This rewinds mag tape zero and then reads each unit and lists the unit identifier message. After the last unit, the tape is rewound and positioned in front of the first unit. No unit has been processed by OLDS control so this command must be followed by a system transfer (ST) to recover control.
FL	Function List – This lists each function ID message in the current unit.
OL	Not allowed. An error flag is output.

1.1.3.5 Transfers

This command causes a controlled transfer to the selected level.

ST System transfer – A system transfer rewinds mag tape zero and simulates a mag tape fill. It reloads the entire system including control.

UT Unit transfer – If a unit transfer is made to a unit other than the current unit, mag tape zero is rewound and each unit is loaded until the selected unit is found and started. If the unit is not found, or the unit is not selected in the unit access word, (see system variables) "ILLEGAL UNIT" will be outputted.

If a unit transfer is made to the current unit, the unit is started by skipping preset code that may change the unit variables according to the system variables. This command is used to restart the current unit, particularly after the unit variables have been changed.

In either case the unit is locked in and will run indefinitely. Only a system transfer or removing the UIW-FIW lock will return automatic control.

FT Function transfer – If a function transfer is made to a function other than the current function, core is searched for the function and the function is started. If the function is not found

"ILLEGAL FUNCTION" will be outputted. The new function will be the only function running. The Unit may be recovered by a unit transfer to the current unit.

If a function transfer is made to the current function, the function is restarted by skipping any preset code that may change the function variables from the system or unit variables is skipped. This command is used to restart the current function particularly after the function variables have been changed.

OxxxxxT	Object Transfer – This sets the machine registers according to the object variables and executes a BRU xxxxx.
OT	Object Transfer – This sets the machine registers according to the object variables and restarts the current object test.
xxxxxxT	Halt and Transfer – This outputs "RTC OFF", turns the Real Time Clock off, sets the machine registers according to the object variables and halts. The halt is followed by a BRU xxxxx.
T	Transfer – This continues the program from the point is was forced to exit.

1.1.3.6 Variables

Each level has optional variables which can be modified. The maximum amount of variables for any level is eight.

SV	Displays System variables
UV	Displays Unit variables
FV	Displays Function variables
OV	Displays Object variables

The variables display may be stopped at any time by toggling Breakpoint 4. This will allow a faster input from the console typewriter to change any variable.

Eleven characters control the changing of any variable.

COMMA (,)	If no numeric characters are entered before a comma, eight dashes will be printed to set the next variable. Any allowable characters inputted before the comma will be added logically to the variable word using 1 + 1 = 0, 1 + 0 = 1. Therefore, bits can be added or deleted.

PERIOD (.)	Replace this variable with the new characters.
SLASH (/)	Input Error-Ignore last input.
0-7	Allowable characters. All other characters will cause an error flag to be printed.

EXAMPLE

```
  UAW       STATUS    UIW/FIW   RADSIZ    DSCSIZ   MEMSIZ    SEED      TIME

67004004   40000000   0000000  20004000  0000400  00000007   36   (BP4 Toggled)

,-------   40004000,,-------   40004000. (carriage return terminates)

                                 -SV

          UAW       STATUS    UIW/FIW    RADSIZ     DS    (BP4 Toggled)

        67004004  00004000  000000000  40004000  00004   (BP4 Toggled)
```

(Carriage return terminates without changes).

Eight variables are displayed at the system level.

UAW Unit Access Word

The 24 bits define 24 possible units that the system can accept. The present assignments are listed in the table of contents.

Unit descriptions can be found in the Table of Contents 1.2 to 1.25.

STATUS	The 24 bits in STATUS define special applications.
	44404404
BIT 0 = 1	Output to line printer address 60 if it is ready. Output to console typewriter for all other cases.
BIT 3 = 1	The computer was operating with enabled interrupts.
BIT 6 = 1	Enable and keep a log of real time clock interrupts.
BIT 12 = 1	Read RAD only to preserve data.
BIT 15 = 1	Read DISC only to preserve data.
BIT 21 = 1	940 definition bit which must stay set.

	UIW/FIW	The first four octals force a pass count for the unit that is running.
		40004000
		If bit 0 is set the unit will loop indefinitely. The last four octals control the cycle count for a function that is running. Octal 4000 or if bit 12 is set the function will not exit.
	RADSIZ	Each channel uses a specific octal to define the amount of rads available. The numbers 0-4 call none, or 1-4 rad cabinets (drums).
		20004000
	BITS 0, 1, 2	Rads present on E channel (UNIT 12 uses this number).
	BITS 3, 4, 5	Rads present on F channel (UNIT 13 uses this number).
	BITS 12, 13, 14	Rads present on W channel (UNIT 15 uses this number).
	BITS 15, 16, 17	Rads present on Y channel (UNIT 16 uses this number).
		No other octals are assigned.
	DSCSIZ	Each channel uses a specific octal to define the disc which is available for testing. Octal 1 implies 8 discs. Octal 2 implies 16 discs. Octal 4 implies 32 discs.
		77007700
	BITS 0, 1, 2	Disc present on E channel (UNIT 18 uses this number).
	BITS 3, 4, 5	Disc present on F channel (UNIT 19 uses this number).
BIT	BITS 12, 13, 14	Disc present on W channel (UNIT 21 uses this number).
	BITS 15, 16, 17	Disc present on Y channel (UNIT 22 uses this number).
		No other octals assigned.

MEMSIZ	The size of the memory is specified by a number 1-4. 00000007 One to four doors or 16K to 64K can be specified.
BITS 21, 22, 23	Memory size in multiples of 16K
SEED	The random number generators in all units use this common location for their seed number. This number will vary as long as CONTROL is in core. When a system is restarted the seed value begins at zero.
TIME	If the Real Time Clock is running, the octal count is held in the TIME cell.

Unit variables will always include the Function Access Word and seven others if needed by the unit. These variables are explained in the Unit Abstract and by their titles.

FAW	Bits 0-23 in this word control access to functions 0-23. A "one" bit allows access.

The eight options for a function are entirely dependent on the particular needs of a function. These variables are explained in the function abstract and by the titles.

The seven object variables displayed are:

AREG	This is the value of the A register at the time any object test transferred to the control mode. It can be changed and will become the new value at transfer back to the test.
BREG	This is the value of the B register.
XREG	This is the value of the X register.
OVRFLO	Bit 0 shows the state of the overflow bit. (Zero or one).
RL1	The relabeling register 0-3 contents at the last setting are held in RL1. If the value is changed, relabeling will change.
RL2	Relabeling register 4-7.

RL4	The monitor relabeling register is normally set to 6, 7 to access the full first 16K. See RL1.
ERRORS	This cell logs the entire system error count.

1.1.3.7 Utilities

Control has two utility functions which can print or modify any core address from 0-177777 by direct addressing.

YxxxxxxP	The print symbol will display the octal contents from the input address xxxxxx until Breakpoint 4 is toggled and aborts the printout. This listing can be sent to the line printer by setting bit zero of the STATUS word to a one. The actual input address is automatically relabeled. A specific count of one through seven words can be displayed before the printout stops if Y is any number greater than zero. The full 8 octals must be entered, i.e., 30000200 will display three words starting at location 200.
xxxxxxM	The modify symbol allows any address (0-64K) to be changed. The rules governing changing variables apply here (see 1.1.3.6. SYNTAX). A maximum of eight sequential addresses can be modified before the M utility transfers back to control. In any case, a carriage return will end the modify mode.

1.1.3.8 Editing

These commands allow the OLDS tape to be edited or added to.

yE xx — Edit – This rewinds mag tapes zero and one. Control is written on mag tape one from core with the current system variables. Successive units are read from mag tape zero, the unit ID message outputted, and then written on mag tape one. When unit xx is read, one of the following is done according to y.

y = 0 Control is returned to the Typewriter.

y = 1 A paper tape fill is simulated.

y = 2 A card fill is simulated.

	The edited units ID message will not be typed until after the continue edit is done.
yC xx	Continue Edit – This is identical to the edit feature except that it continues from its current position. It starts by outputting the current unit ID message and then writing the current unit on mag tape one. xx is the next unit to be edited.
B	Back Space – Mag Tape zero is backspaced one unit.
E 24	Copy All – This command copies an OLDS tape putting in the current system variables.

1.1.4 BREAKPOINT OPTIONS

The following breakpoint options which are tested in OLDS will be fixed for all 940 diagnostic programs. Individual diagnostic programs are not to include any breakpoint tests:

BPT1	Set – Loop on current object test. Reset – Proceed to next object test.
BPT2	Set – Do not proceed to control on errors. Reset – Proceed to control on errors.
BPT3	Set – Inhibit error messages. Reset – Output error message on selected peripheral devices.
BPT4	Toggle – Go to control and/or terminate output (BPT-1, 2, 3 are ignored). Not Toggled – Normal operations (BPT-1, 2, 3 are tested).

To proceed to control from an output loop, set BPT3, then set BPT4, reset BPT3 and wait one second. When BPT4 is reset, control will be accessed.

1.1.5 OPERATING PROCEDURES

1.1.5.1 Loading the System

The binary tape loads from the "W" channel, using only a "WIM" (W to memory) and a "BRU" (Branch) instruction. The explanation of the loader can be found in Section 1.1.5.8. The magnetic tape unit address is zero.

Memory clear, set run, and fill from mag tape. Control will read into locations 30 through 4000 and set operation for automatic running. Each unit will then be read in, operated, and dismissed until the finish of the last unit.

Tape will then rewind and begin with the first unit after by-passing control to preserve any variable changes.

1.1.5.2 Manual Starting

If typewriter control is required at load time, breakpoint four should be set before loading the system.

The control program will load, print the control dash, and wait for input from the typewriter. Any control functions can be used at this time. Usually the system variables are adjusted to conform to the particular system being tested. Automatic control will return if a T is input. If a UXXT is input the XX unit will lock in and run. Function transfers will also lock in the function. See Section 1.1.3.5 TRANSFERS.

1.1.5.3 Initializing

System variables can be changed at any time control to the typewriter is requested by toggling BP4. The SV directive does this.

To change unit variables, the unit must have been loaded before the variables can be accessed by the UV directive. If the unit is dismissed, the variables will be reset according to the system configuration. Any changes will have to be re-entered. This condition is also valid for the function and object level.

Usually, the particular unit or function is locked in by a forced transfer. Since this condition disallows any dismissal, the variables can be changed at any time.

1.1.5.4 Independent Operation

If the UIW/FIW lock is set any unit or function will not exist except for transfers out of the unit or function. In this case, the new unit or function is still locked. An object test is locked by breakpoint one.

A typical typewriter control could be:

```
U-0    ---------------    Unit ID's which were printed during
                          auto-run control
U-2    ---------------                 ↑
                                       |
U-12   ---------------                 |
                                       ↓
U-23   ---------------    Unit ID's which were printed during
                          auto-run control
U-0    ---------------
```

–SV	BP4 toggled called control
______UIW/FIX______	System variables
______00000000______	requested.
–U0T	UNIT zero transfer or transfer to same unit
–U2T	BP4 toggled and U2 selected
U–2 --------	UNIT ID, MESSAGE
–SV	BP4 toggled and variables requested.
______UIW/FIW______	
______40000000______	Clear unit
______00000000______	Iteration Word
–T	
U–12 -------------	Automatic
U–23 -------------	Operation proceeds
U–0 -------------	

Any unit, function, or object can be run as an individual program. Usually, an error stop at control determines whether a loop condition is necessary, but this predetermined loop can be set at any time. The loop necessary to duplicate any process cannot be readily defined, but if the variables, and the identifiers are printed, some decision can be made as to the next step. Perhaps obtaining the function listing (FL) would be beneficial in finding an alternate subprogram to analyze the error results.

The print feature (xxxxxxP) can also be useful to list the failing object test for study of the micro instructions.

1.1.5.5 Error Procedure

When an error is located by a program, information is displayed and control is returned to the keyboard. Information displayed by a diagnostic may include the logic equation involved. Location of possible bad module(s), module type, or signal name(s). An exerciser will display the error and will give information pertinent to location of the problem.

At this time, the operator has several options. He may identify the unit function, or object test by typing UI, FI, or OI respectively. OI will print the core location of the object test, identifying the start of the last object test. By typing UV, FV, or OV, the current variables will be displayed. Object variables (OV) include contents of central registers, the status of relabeling registers and overflow, and the error count at the time the error was discovered.

If the operator wishes to loop on the failing object test, he may set Breakpoint one (BP1) and type OT. Setting Breakpoint two (BP2) will inhibit the error halt, and setting Breakpoint three (BP3) will inhibit the error typeout. Keyboard control may be regained by toggling Breakpoint four (BP4).

If the operator wishes to continue from where the error was detected without looping, he types T with BP1 reset. Again, the setting of BP2 and BP3 will determine whether or not to halt on or report errors. If T is typed and BP1 is set, the program will loop on the current object test.

If the operator suspects that he is making an operating error, he may request a unit, function, or object abstract by typing UA, FA, or OA respectively. Information pertinent to operation at the various levels is included in the abstract.

1.1.5.6 Copying

Any 940/930 with two mag tape units may be used to copy the OLDS tape.

Mount the source tape on mag tape zero. Set BP4 and execute a mag tape fill. When the tape stops, reset BP4. Control will be returned to the console typewriter.

If it is wished to change the system variables do so now as explained in Section 1.1.3.6.

Mount the new tape on mag tape one. Be sure it has a file protect ring installed (not file protected).

Type E24 (CR), both tapes will rewind and the copying will begin. The Unit ID message will be output after each unit is read and before it is written.

Wait until both tapes have stopped and control is returned to the console typewriter.

The new tape is now complete and may be checked.

1.1.5.7 Editing and Checking

Editing may be done on any 940/930 with two mag tapes and a card or paper tape reader for the edit source.

Mount the source OLDS tape on mag tape zero. Set BP4 and execute a mag tape fill. When the tape stops release BP4, control will return to the console typewriter.

If the system variables are to be changed, do so now as described in Section 1.1.3.6.

Mount the new tape on mag tape one. Be sure it has a file protect ring (not file protected).

To begin an edit, type yE xx (CR), where y is the edit source and xx is the unit to be edited. Both tapes are rewound and a control record is written from core on mag tape one and successive units are copied from mag tape zero to mag tape one until unit xx has been read. As each unit is written on mag tape one, the unit ID is output.

y specifies the edit source where:

y = 0	control is returned to the console typewriter.
y = 1	a paper tape fill is simulated.
y = 2	a card fill is simulated.

If y = 0, the unit may be edited using the modify command.

If y = 1 or 2, the new unit is read in and control is returned to the console typewriter.

To continue, type yC xx (CR), where y is the edit source and xx is the next unit to be edited. This operates identically to the E directive above except that the tapes are not rewound and the control record is not generated.

To insert a new unit use either an E xx (CR) or C xx (CR) where xx is the unit number the new unit is to follow. Control will return to the console typewriter. Now type B; this will backspace mag tape zero. Now manually execute a card or paper tape fill as desired which will read in the next unit.

Control will again be returned to the console typewriter. Now the edit may be continued or terminated as desired.

Type C30(CR) to complete the edit function. Wait until both tapes have stopped moving and the control is returned to the console typewriter. The new tape is now complete.

To check the new tape, remove the file protect ring from the tape on mag tape one. Reset mag tape one to be mag tape zero. Force the other tape unit to the NOT READY condition. Type UL. The units will be read and each unit identifier will be printed on the console typewriter. When the last unit is read, control will return to the console typewriter. The tape will rewind and position in front of the first unit. Any re-reads or an unsuccessful cycle will mean a bad or marginal tape. A new tape should be made.

1.1.5.8 The Loader

The format of the code on the OLDS tape is a WIM to an address followed by the code word to be put at the address. This motif was used because it taxes the input channel least. Therefore, there is a greater probability of successfully loading a unit program under adverse conditions.

Any tape record has the following format:

```
WIM   3

WIM   4              loader

BRU   2

WIM   xxxxx

DATA                 program record

WIM   xxxxx+1

DATA

  -

  -

  -

  -

BRU   START
```

When a fill is executed, the hardware forced WIM 2 will read the WIM 3 off the tape into location 2 and step to location 2. The WIM 3 will read the WIM 4 off the tape to location 3 and step to location 3. The WIM 4 will now read the BRU 2 to location 4, and step to location 4. The BRU 2 will return the control to location 2 and the WIM 3 will read the next tape word. This word is a WIM xxxxx. The next tape word is the data to be stored at xxxxx. Through the remainder of the tape block, a WIM xxxxx will be read previous to the data to be put at that location. The last tape word is a BRU START which goes to location 3 and causes the loader to branch to the starting location of the loaded program.

1.1.6 INTERFACE REQUIREMENTS

The following interface specifications must be strictly observed by all 940 diagnostic programs.

1.1.6.1 Unit Level Interface

All programs will start with ORG 4000 or BSS 4000. The first two executable instructions of a Unit program must be:

```
BRM UNIT

NOP UPT
```

where UPT is a label, selected by the User, pointing to the beginning of a Unit Parameter Table.

The last instruction of a Unit program must be BRM DONE.

The Unit Parameter Table and Unit Variable Table will have the following format. The label and operand names can be chosen by each programmer.

```
*
*   UNIT PARAMETER TABLE
*
                                        (octal number)

1)  UPT  NOP    UIM        Unit identifier message address

2)       NOP    UAM        Unit abstract message address

3)       NOP    UVM        Unit variables message address

4)       TWO    UVT        Unit variables display control word
*
*   UNIT VARIABLES TABLE
*
5)  UVT  DATA  (octal number)  Unit identifier

6)       DATA  (octal number)  Function access word
```

Where ONE and TWO are OPD defined as:

```
ONE   OPD   01000000

TWO   OPD   02000000
```

1.1.6.2 Definition of Unit Parameters

Entry one. The operand field of entry one UIM, must point to a Unit Identifier Message.

```
Example – UIM BCD        ' UNIT IDENTIFIER " '
```

Entry two. The operand field of entry two – UAM, must point to a Unit Abstract Message which will describe the Unit's use.

```
Example – UAM BCD        ' UNIT ABSTRACT " '
```

Entry three. The operand field of entry three – UVM, must point to a Unit Variable Message, which defines the Unit variables. It must be in a format compatible with the variables to be displayed directly below it.

```
Example – UVM BCD        ' FAW UNIT  ID " '
```

Entry four. This is a control word used to display the Unit variables.

The op-code field, bits 3 to 8, must contain the number of variables to be displayed, not to exceed eight (10_8).

The operand field will point to the Unit Variables Table.

1.1.6.3 Definition of Unit Variables

Entry five. The contents of UVT is an octal word, with a bit set, corresponding to the Unit identifier number--i.e., Unit No. 3 is 04000000. The bit position will correspond to the Unit number.

Entry six. The operand field of entry six is a Function access word, which controls the access of the various Functions contained in the Unit.

Only those Functions, whose Function Identifier corresponds to a bit set in this word, will be accessed by CONTRL.

The Function access word is set-up such that bit zero will correspond to Function zero, etc.

The Functions will be accessed sequentially by CONTRL, from the most significant to the least significant bit position.

1.1.6.4 Function Level Interface

Each Function must observe the following interface requirements:

The first two executable instructions of a Function must be

```
BRM     FUNCTION

NOP     FPT1
```

where, FPT1 points to the Function Parameter Table.

The label FPT1 is not fixed and may vary for different Functions.

The last instruction of a Function must be BRM FDONE.

The Function Parameter Table must have the following format. Label and operand names are not fixed and may be varied for each function.

```
*
*   FUNCTION PARAMETER TABLE 1
*

1)  FPT1   NOP    F1M1     Function Identifier Message Address

2)         NOP    FAM1     Function Abstract Message Address

3)         NOP    FVM1     Function Variables Message Address

4)         ONE    FVT1     Function Display Control Word

5)         PZE    FUNC2    Address of Next Function

6)         DATA   (Octal)  Function Identifier Number
```

where ONE is OPD defined.

1.1.6.5 Definition of Function Parameters

Entry one. The operand field of entry one – F1M1, will point to a Function identifier message.

Entry two. The operand field of entry two – FAM1, will point to a Function abstract message. This message will describe the Function's use.

Entry three. The operand field of entry three – FVM1, will point to a Function variable message, which defines the Function variables. It must be in a format compatible with the variables to be displayed directly below it.

Entry four is a control word for the display of Function variables. The op-code field, bits 3 to 8 must contain the number of variables to be displayed. The operand field will contain an address or label pointing to the Function Variables Table.

Entry five. This parameter is a link to the next Function. The op-code field will contain a PZE. The operand field will contain an address or label pointing to the next Function.

The contents of FPT1 is an octal word with a bit set, corresponding to the Function identifier number, i.e., Function number 1 is 20000000.

1.1.6.6 Function Variables Table

The format of the Function Variables Table is up to the programmer, correlation must be maintained between the Function Variables Table and the pointers in the Function Parameters Table which point to the entries in the Function Variables Table. Also, there can be no more than eight (8) entries.

The FUNCTN subroutine, in OLDS, moves the parameters in a System storage area and pre-sets all changeable interrupt BRU's to the normal state (BRU DIVERT).

Any Function Variables which are dependent on system configuration such as memory size, device size of optional hardware must be constructed from the System variables and stored in the Function Variables Table prior to accessing the first Object program within the Function.

1.1.6.7 Object Level Interface

Each Object program in a Function must observe the following interface requirements:

The first instruction in an Object program must be,

```
BRM OBJECT
```

OBJECT will test breakpoint 4, one of only two means of entering CONTRL.

The last instruction of an Object program must be,

```
BRM END
```

END will test breakpoint 1 and either repeat the current Object program or proceed to the next Object program.

If interrupts, traps, or POP's are anticipated, the following instructions are required to be set up before they occur:

```
BRM RETURN

NOP INT
```

where INT is a return address for the interrupt which is serviced in an Object program. Traps and POP's are serviced similarly.

The BRM RETURN will direct the response of an interrupt, trap or POP to one or more return addresses specified in the operand field of the NOP instructions.

If an Object program detects an error, the following instructions are required to process the error,

```
BRM   ERROR

NOP  ERMSG
```

where ERMSG points to an error message.

All error messages will use the BCD directive and terminate with a " (37_8) symbol.

The following example illustrates the format of a typical Object program,

```
*

*                  OBJECT PROGRAM

*

OBJCT1             BRM      OBJECT

                   BRM      RETURN

                   NOP      TRAP
     .
                   CLA

                   STA      RL1

                   STA      RL2

                   EOM      020400

                   POT      RL1               OBJECT

                   EOM      021000            PROGRAM
```

```
                  POT     RL2
                  STA     0, 4
                  BRM     ERROR
                  NOP     ERMSG1
                  BRU     FINISH
     TRAP         LDA     DIVERT
                  EOR     =ROT
                  SKA     =37777
                  BRU     *+2
-                 BRM     ERROR
-                 NOP     ERMSG2
-    FINISH       BRM     END
```

where the dash (-) indicates the pertinent interface information.

1.1.6.8 The Use of "ERROR" and "REPORT" Messages

Messages may be outputted through REPORT which does not interface CONTRL, and ERROR which does.

These rules apply to both.

All message pointers must have NOP's in the operand field;

```
     NOP  ERRMSG
```

Messages may be linked together by the presence of a bit in position zero.

```
     NOP  MSGS1, 4
     NOP  MSGS2, 4
     NOP  MSGS3.
```

The contents of registers A, B, X and overflow may be outputted by setting Bit 1;

```
NOP MSGS1, 4

NOP MSGS2, 2
```

A message which outputs the registers must be by itself or the last message in a linked chain.

1.1.6.9 Other Interface Requirements

Users may use only the following System subroutines:

UNIT

FUNCTN

OBJECT

DIVERT

RETURN

ERROR

END

FDONE

DONE

REPORT

Under no circumstance will a user use any other System subroutine.

All messages will use a BCD directive and terminate with a (")(37_8) symbol.

All labels required to interface between OLDS and the various Program levels are to be defined by EQU directives. This will eliminate all uses of external references and definitions.

1.1.6.10 List of Major Subroutines

OLDS is composed of the following subroutines all designed to perform specific functions:

CONTRL	This sub-program controls all operator input to OLDS and its component diagnostic programs as well as handling most

	of the output from OLDS. This routine is entered through OBJECT or ERROR.
UNIT	A subroutine to pick-up all required parameters and variables necessary to define a specific Unit level program and interfaces the currently loaded user program to the system.
DONE	A subroutine which determines if a Unit is to be dismissed or not.
FUNCTN	A subroutine to pick-up all required parameters and variables necessary to define a specific function level program.
FDONE	A subroutine which terminates a function.
OBJECT	A subroutine to mark the starting address of the last object program to be initiated. This routine contains the only linkage to control and processes the 'return to control' option.
END	A subroutine which processes the tight loop option.
RETURN	A subroutine that maintains address linkages to the current object program for all POP, trap and interrupt receivers.
DIVERT	A subroutine that marks the locations where interrupts, traps and POP's are received.
ERROR	This subroutine is the common error reporting routine used by all 940 diagnostic programs. The "inhibit error output and error halt options" are processed by this routine.
REPORT	Common output handler used by all 940 diagnostic programs.

1.2 UNIT 0 940 CPU DIAGNOSTIC AND PRETEST

1.2.1 Pretest

The purpose of PRETEST is to verify the correct operation of all the instructions used by the executive. The following instructions are tested in the 930 mode:

LDA	LDB	LDX			
STA	STB	STX			
BRM	BRR				
SKA	SKB	SKE	SKM	SKN	SKS
ETR	EOR	MRG			
MIN					
LCY 2	RCY 1				
MIW (To teletype)					

It is assumed that the following instructions and operations operate without errors:

WIM

NOP

BRU

HLT

EOM (To device and buffer)

SKS (To device and buffer)

Manual Register Exchange

Breakpoint Tests

PRETEST will not attempt to diagnose failures. If an error is detected, the program will come to a halt.

Breakpoints are utilized as follows:

BP1	Set – Loop on current test Reset – Continue to next test

BP2	Set – Do not halt on errors Reset – Halt on errors
BP3	Set – Loop on PRESET Reset – Perform WIM Test, Monitor Map Test, return to the executive and load first Unit from the diagnostic tape.

At errors, the halts will indicate as follows:

LOCATION	CAUSE
04017	SKS skipped
04027	SKE didn't skip
04037	SKE didn't skip
04047	SKE didn't skip
04060	SKE skipped
04071	SKE skipped
04102	SKE skipped
04113	SKE skipped
04124	SKE skipped
04135	SKE skipped
04145	SKE didn't skip
04156	SKE didn't skip
04165	A error during SKE
04172	B error during SKE
04177	X error during SKE
04210	SKA didn't skip
04217	A error during SKA
04224	B error during SKA
04231	X error during SKA
04242	SKA didn't skip
04251	A error during SKA
04256	B error during SKA
04263	X error during SKA
04275	SKA skipped
04304	A error during SKA
04311	B error during SKA
04316	X error during SKA
04330	SKA skipped
04337	A error during SKA
04342	B error during SKA
04351	X error during SKA
04363	SKA skipped
04372	A error during SKA
04377	B error during SKA
04404	X error during SKA
04415	SKM didn't skip
04424	A error during SKM

LOCATION	CAUSE
04431	B error during SKM
04436	X error during SKM
04447	SKM didn't skip
04456	A error during SKM
04463	B error during SKM
04470	X error during SKM
04502	SKM skipped
04511	A error during SKM
04516	B error during SKM
04523	X error during SKM
04535	SKM skipped
04542	A error during SKM
04551	B error during SKM
04556	X error during SKM
04570	SKM skipped
04577	A error during SKM
04604	B error during SKM
04611	X error during SKM
04622	SKN didn't skip
04631	A error during SKN
04636	B error during SKN
04643	X error during SKN
04655	SKN skipped
04664	A error during SKN
04671	B error during SKN
04676	X error during SKN
04707	SKB didn't skip
04716	A error during SKB
04725	B error during SKB
04730	X error during SKB
04741	SKB didn't skip
04750	A error during SKB
04755	B error during SKB
04762	X error during SKB
04774	SKB skipped
05003	A error during SKB
05010	B error during SKB
05015	X error during SKB
05027	SKB skipped
05036	A error during SKB
05043	B error during SKB
05050	X error during SKB
05062	SKB skipped
05071	A error during SKB
05076	B error during SKB
05101	X error during SKB
05120	A error during ETR

LOCATION	CAUSE
05125	B error during ETR
05132	X error during ETR
05147	A error during EOR
05154	B error during EOR
05161	X error during EOR
05176	A error during MRG
05203	B error during MRG
05210	X error during MRG
05236	A error during MIN
05245	B error during MIN
05250	X error during MIN
05276	A error during MIN
05303	B error during MIN
05310	X error during MIN
05336	A error during MIN
05343	B error during MIN
05350	X error during MIN
05376	A error during MIN
05403	B error during MIN
05410	X error during MIN
05425	A error during RCY
05432	B error during RCY
05437	X error during RCY
05454	A error during RCY
05461	B error during RCY
05466	X error during RCY
05503	A error during RCY
05510	B error during RCY
05515	X error during RCY
05532	A error during RCY
05537	B error during RCY
05542	X error during RCY
05561	A error during LCY
05566	B error during LCY
05573	X error during LCY
05610	A error during LCY
05615	B error during LCY
05622	X error during LCY
05637	A error during LCY
05644	B error during LCY
05651	X error during LCY
05666	A error during LCY
05673	B error during LCY
05700	X error during LCY
05717	MARK error during BRM
05727	A error during BRM
05734	B error during BRM

LOCATION	CAUSE
05741	X error during BRM
05752	BRR went to MARK instead of MARK+1
05760	BRR clobbered MARK
05770	A error during BRR
05775	B error during BRR

1.2.2 Function 0 - 940 Monitor Mode CPU Diagnostic

This function locates failures in the 940 Monitor Mode and gives module locations of the probable fault.

1.2.3 Function 1 - 940 User Map Diagnostic

This function locates failures in the User Map and the Memory Trap logic and gives module locations of the probable fault. No use of upper memory is made and the diagnostic will operate with only 16K of memory present.

1.2.4 Function 2 - 940 Monitor/User Transition Diagnostic

This function locates failures in the transitions between user and monitor modes. It also checks the monitor to user transition trap logic. It gives module locations of the probable fault.

1.2.5 Function 3 - 940 Privileged Instruction Diagnostic

This function locates failures in the privileged instruction logic and gives module locations of the probable fault.

1.2.6 Function 4 - 940 Real Time Clock Diagnostic

This function locates failures in the RTC and checks the time against the computer clock ±10% and gives module locations of the probable fault.

1.3 UNIT 1 - 940 CPU EXERCISER

This unit contains the exercisers and special functions devoted to the CPU. There is only one Unit Variable, FAW, which contains bits corresponding to the functions to be activated. Normally, Functions 0, 1, 2 and 3 are automatically activated.

1.3.1 Function 0 - General Instruction Exerciser

This function is based on the 930 Instruction Examiner and operates on data blocks in the same manner. It presets the registers, the operand, and overflow, executes the instruction, and then checks for skip, registers, operand, and overflow.

At an error, the message will tell the type of error result, and the expected result.

There are six Function Variables:

CREG	This is the initial operand
AREG	This is the initial A Register
BREG	This is the initial B Register
XREG	This is the initial X Register
FIW	This is the number of passes per function
INST	This is the instruction under test; if the error occurs during a multiple or divide with BP2 reset, lock the object test in by setting BP1; from CONTROL do a F 22T for a multiply or a F 23T for a divide. This will type out the function identifier of the analyzer, type T to get a breakdown of the problem by clock times. Do a F 0T and the program will return to the same test case for trouble shooting.

1.3.2 Function 1 - 940 Add Exerciser

This function generates random adds and compares them against the result of a simulated add. It generates random numbers for A, B, X and operand, executes the add, and compares the registers and overflow against a simulated add which uses only the logical instructions.

At an error, the message will tell the type of error, the error result, and the expected result.

There are six Function Variables:

SEED	The starter for the random number generator
CREG	The initial operand
AREG	The initial A Register
BREG	The initial B Register
XREG	The initial X Register
FIW	The number of passes per function

A specific ADD may be tested by:

Do a F 1T and toggle BP4 to reach CONTROL, preset the Function Variables for the add to be tested, set BP1 to lock that test in, and type a T. The add exerciser will then be locked in on that add.

A specific random number sequence may be run again:

Do a F 1T and toggle BP4 to reach CONTROL, set the starting SEED in the Function Variables.

Do a F 1T. The exerciser is now running through a random number sequence determined by the SEED.

1.3.3 Function 2 - 940 Shift Exerciser

This function generates random shifts and compares them against simulated shifts in the same manner as the Function 1.

Error output and the function variables are the same as Function 1.

1.3.4 Function 3 - 940 Multiply Exerciser

This function generates random multiplies and compares them against a simulated multiply in the same manner as the add exerciser.

Error output and the Function Variables are the same as Function 1.

Specific multiplies or random sequences may be initiated in similar manner to Function 1.

In addition, if any error occurs, a breakdown of the multiply by computer clocks times may be printed. Lock the test in by setting BP1.

Reach CONTROL either by BP4 or an error stop. Do a F 22T, this will type the function identifier message. Type T for a breakdown of the test case.

Set BP1, do a F 2T, and the program will return to the same test.

1.3.5 Function 4 - 940 Divide Exerciser

This function generates random divides and compares them against a simulated divide in the same manner as the add exerciser. Error output and the Function Variables are the same as Function 1.

Specific divides or random sequences may be initiated in similar manner as Function 1.

In addition, if an error occurs, a breakdown of the Divide by computer clock times may be printed.

Lock the test in by setting BP1.

Reach CONTROL either by BP4 or an error stop. Do a F 23T. This will type out the function identifier message. Type T to get a breakdown of the test case. Set BP1, do a F 3T, and the program will return to the same test.

1.3.23 Function 22 - Multiply Analyzer

This function simulates the steps of a multiply and prints the results at each computer clock time. It is the same simulator used by the Multiply Exerciser. No use of the adder or the right shift adder is made. Only logical, and register exchange instructions are used.

This function may be used with Function 0 or Function 2 as described above. In addition, specific multiplies may be analyzed.

Toggle BP4 to get to CONTROL.

Do a F 22T, setting BP4, the function identifier message will be typed.

Set up the Function Variables.

Type T, the program will display the multiply steps, and return to CONTROL.

1.3.24 Function 23 - Divide Analyzer

This function simulates the steps of a divide and prints the results at each computer clock time. It is the same simulator used by the Divide Exerciser.

No use of the adder or the right shift adder is made. Only logical and register exchange instructions are used.

This function may be used with Function 0 or Function 3 as described above. In addition, specific divides may be analyzed.

Toggle BP4 to get to CONTROL.

Do a F 23T. The function identifier message will be typed.

Set up the Function Variables.

Type T, the program will display the divide steps, and return to CONTROL.

1.4 UNIT 2 - FLOATING POINT ARITHMETIC UNIT FPAU

The diagnostic is similar to other diagnostics under the OLDS system. It will give the probable locations of module failures.

The exercisers generate random test cases and compare the result of the FPAU operations with simulated results. If an error occurs, the actual and expected results are printed.

The special case exerciser checks particular cases such as divide by zero.

The simulators give the correct result of any specified test case. They can also give the correct state of the FPAU registers at any clock time.

The utilities give the actual result of any specified test case. They can also give the actual state of the FPAU registers at any clock time.

If an error occurs in any exerciser, do a function transfer to the appropriate simulator to get the correct result. The failing test case is already preset in the function variables.

Then do a function transfer to the appropriate utility to get the actual result. Executing a FL directive will list the identifiers of the various simulators. The failing case is still preset in the function variables.

To return to the exerciser, set BP1 to loop, set BP2 and BP3 to suppress error halt and error typeout. Do a function transfer back to the exerciser. The exerciser will loop on the failing test case and the FPAU may be examined.

1.4.1 Function 0 - FPAU Diagnostic

1.4.2 Function 1 - Add Exerciser

This function generates random add test cases and compares the results with a simulator.

If a failure occurs, the actual and the expected FPAU may be displayed by doing a function transfer to the add utility. The correct steps may be displayed by doing a function transfer to the add simulator.

Specific Test cases may be run.

1. Toggle BP4 to reach control.
2. Set the function variables to the desired values.
3. Set BP1 to loop in the test case.
4. Type T.

The specified test case is now being run.

1.4.3 Function 2 - Subtract Exerciser

This function generates random subtract test cases and compares the result with a simulator.

If a failure occurs, the actual and the expected result will be given. The intermediate steps of the FPAU may be displayed by doing a function transfer to the subtract utility. The correct steps may be displayed by doing a function transfer to the subtract simulator.

Specific test cases may be run.

1. Toggle BP4 to reach control.
2. Set the function variables to the desired values.
3. Set BP1 to loop in the test case.
4. Type T.

The specified test case is now being run.

1.4.4 Function 3 - Inverse Subtract Exerciser

This function generates random inverse subtract test cases and compares the result with a simulator.

If a failure occurs, the actual and the expected result will be given. The intermediate steps of the FPAU may be displayed by doing a function transfer to the inverse subtract utility. The correct steps may be displayed by doing a function transfer to the inverse subtract simulator.

Specific test cases may be run.

1. Toggle BP4 to reach control.
2. Set the function variables to the desired values.
3. Set BP1 to loop in the test case.
4. Type T.

The specified test case is now being run.

1.4.5 Function 4 - Multiply Exerciser

This function generates random multiply test cases and compares the result with a simulator.

If a failure occurs, the actual and the expected result will be given. The intermediate steps of the FPAU may be displayed by doing a function transfer to the multiply utility. The correct steps may be displayed by doing a function transfer to the multiply simulator.

Specific test cases may be run.

1. Toggle BP4 to reach control.
2. Set the function variables to the desired values.
3. Set BP1 to loop in the test case.
4. Type T.

The specified test case is now being run.

1.4.6 Function 5 - Divide Exerciser

This function generates random divide test cases and compares the result with a simulator.

If a failure occurs, the actual and the expected result will be given. The correct steps may be displayed by doing a function transfer to the divide simulator.

Specific test cases may be run.

1. Toggle BP4 to reach control.
2. Set the function variables to the desired values.
3. Set BP1 to loop in the test case.
4. Type T.

The specified test case is now being run.

1.4.7 Function 6 - Inverse Divide Exerciser

This function generates random inverse divide test cases and compares the result with a simulator.

If a failure occurs, the actual and the expected result will be given. The intermediate steps of the FPAU may be displayed by doing a function transfer to the inverse divide utility. The correct steps may be displayed by doing a function transfer to the inverse divide simulator.

Specific test cases may be run.

1. Toggle BP4 to reach control.

2. Set the function variables to the desired values.

3. Set BP1 to loop in the test case.

4. Type T.

The specified test case is now being run.

1.4.8 Function 7 - Special Case Exerciser

This function checks special arithmetic cases such as division by 0. It operates on data blocks in the same way the CPU instruction examiner does. The function variables give the op-code and operands. This function may be used with the utility and simulator functions in the same way as the other exercisers.

1.4.13 Function 12 - Add Simulator

This function allows add cases to be simulated and the intermediate steps displayed.

The function variable format controls the display:

0	=	Display result only
1	=	Display odd clock times only
2	=	Display even clock times only
3	=	Display all clock times

Refer to the section 1.4 for use of this function with the exerciser.

1.4.14 Function 13 - Subtract Simulator

This function allows subtract cases to be simulated and the intermediate steps displayed. The function variable format controls the display:

0	=	Display results only
1	=	Display odd clock times only
2	=	Display even clock times only
3	=	Display all clock times

Refer to the section 1.4 for use of this function with the exerciser.

1.4.15 Function 14 - Inverse Subtract Simulator

This function allows inverse subtract cases to be simulated and the intermediate steps displayed.

The function variable format controls the display:

0	=	Display result only
1	=	Display odd clock times only
2	=	Display even clock times only
3	=	Display all clock times

Refer to the section 1.4 for use of this function with the exerciser.

1.4.16 Function 15 - Multiply Simulator

This function allows multiply cases to be simulated and the intermediate steps displayed. The function variable format controls the display:

0	=	Display result only
1	=	Display odd clock times only
2	=	Display even clock times only
3	=	Display all clock times

Refer to the section 1.4 for use of this function with the exerciser.

1.4.17 Function 16 - Divide Simulator

This function allows divide cases to be simulated and the intermediate steps displayed. The function variable format controls the display:

0	=	Display result only
1	=	Display odd clock times only
2	=	Display even clock times only
3	=	Display all clock times

Refer to the section 1.4 for use of this function with the exerciser.

1.4.18 Function 17 - Inverse Divide Simulator

This function allows inverse divide cases to be simulated and the intermediate steps displayed. The function variable format controls the display:

0	=	Display result only
1	=	Display odd clock times only
2	=	Display even clock times only
3	=	Display all clock times

Refer to section 1.4 for use of this function with the exerciser.

1.4.19 Function 18 - Add Utility

This function allows add cases to be presented to the FPAU. Format is the display control word.

0	=	Display result only
1	=	Display odd clock times
2	=	Display even clock times
3	=	Display all clock times

Refer to section 1.4 for the use of this function with the add exerciser.

1.4.20 Function 19 - Subtract Utility

This function allows subtract cases to be presented to the FPAU. Format is the display control word.

0	=	Display result only
1	=	Display odd clock times
2	=	Display even clock times
3	=	Display all clock times

Refer to section 1.4 for the use of this function with the subtract exerciser.

1.4.21 Function 20 - Inverse Subtract Utility

This function allows inverse subtract cases to be presented to the FPAU. Format is the display control word.

0	=	Display result only
1	=	Display odd clock times
2	=	Display even clock times
3	=	Display all clock times

Refer to section 1.4 for the use of this function with the inverse subtract exerciser.

1.4.22 Function 21 - Multiply Utility

This function allows multiply cases to be presented to the FPAU. Format is the display control word.

0	=	Display result only
1	=	Display odd clock times
2	=	Display even clock times
3	=	Display all clock times

Refer to section 1.4 for the use of this function with the multiply exerciser.

1.4.23 Function 22 - Divide Utility

This function allows divide cases to be presented to the FPAU. Format is the display control word.

0	=	Display result only
1	=	Display odd clock times
2	=	Display even clock times
3	=	Display all clock times

Refer to section 1.4 for the use of this function with the divide exerciser.

1.4.24 Function 23 - Inverse Divide Utility

This function allows inverse divide cases to be presented to the FPAU. Format is the display control word.

0	=	Display result only
1	=	Display odd clock times
2	=	Display even clock times
3	=	Display all clock times

Refer to section 1.4 for the use of this function with the inverse divide exerciser.

1.5 UNIT 3 - 940 MEMORY DIAGNOSTIC AND EXERCISER FOR THE 2ND 16K

This unit contains the diagnostics and exercisers which are necessary to check addresses 40000 to 77777 octal. There is only one Unit Variable, FAW, which contains bits corresponding to the function to be activated. Normally Functions 1, 2, 3 and 4 are automatically activated. Operation in the non-interleaved mode is necessary for correct diagnosis.

1.5.2 Function 1 - 940 Map Diagnostic

This function diagnoses fault in the User Map, Monitor Map, and the Memory Traps. The diagnosis does not depend on the correct operation of any upper memory. Error messages will describe which module locations are associated with the probable fault and the signal names involved. These are all in the CPU.

There are no Function Variables.

1.5.3 Function 2 - 940 M-Register Diagnostic

This function checks that ones and zeros can be stored in all four quadrants of the memory under test. Error messages will describe which module locations are associated with the probable fault, and the signal names involved. These are all in the memory. There are no Function Variables.

1.5.4 Function 3 - 940 Address Driver Diagnostic

This function spreads addresses through all upper core, and checks specific addresses to determine address drive and sink module failures. Error messages will describe module locations that are associated with the probable fault, and the signal names involved. These are all in the memory.

There are no Function Variables.

1.5.5 Function 4 - 940 Memory Noise Test

This function generates worst case noise patterns and access histories. This function is suitable for a memory "SHMOO".

Error messages will state either parity error and give the location involved, or memory noise error and give the error word, the expected word, and the address.

There are no Function Variables.

1.5.6 Function 5 - 940 Memory Signal Bouncer

This function is a Scope Aid. It causes all RL flip-flops, all M flip-flops, all L lines, and all SEL lines to bounce, for an aid in signal tracing.

There are no Function Variables or error routines.

1.6 UNIT 4 - 940 MEMORY DIAGNOSTIC AND EXERCISER FOR THE 3RD 16K

This unit is similar to Unit 3. (Refer to all Functions in Unit 3.)

1.7 UNIT 4 - 940 MEMORY DIAGNOSTIC AND EXERCISER FOR THE 4TH 16K

This unit is similar to Unit 3. (Refer to all Functions in Unit 3.)

1.8 UNIT 6

Not assigned.

1.9 UNIT 7

Not assigned.

1.10 UNIT 8

Not assigned.

1.11 UNIT 9

Not assigned.

1.12 UNIT 10

Not assigned.

1.13 UNIT 11

Not assigned.

1.14 UNIT 12 - E CHANNEL AND 9367 RAD TEST

Unit 12 includes a failure analysis for E channel with one RAD controller and four mechanical RAD. The error outputs are relative to the channel or RAD, depending on the particular function which is in process. Signal names and probable modules which could cause the response are printed in the error messages.

1.14.1 Function 0

Not assigned.

1.14.2 Function 1 - E Channel Test

The channel is exercised without benefit of an I/O device to attempt an analysis of faults particular to the channel only. No Function Variables are used.

1.14.3 Function 2 - RAD Primary Test for RAD Addresses 00000 to 17777

This function checks basic responses from the RAD controller and its link to the first selector unit. Error listings will call the SEL Unit if the probable cause is in that section of the RAD. There are no Function Variables.

1.14.4 Function 3 - RAD Addresses 20000 to 37777

The second RAD select unit is tested in this function with similar error reports as defined in Function two.

1.14.5 Function 4 - RAD Addresses 40000 to 57777

Function four operates similar to Function two.

1.14.6 Function 5 - RAD Addresses 60000 to 77777

Function five operates similar to Function two.

1.14.7 Function 6 - RAD Head Analysis for RAD One

This function tests each head access for proper addressing and correct data transfer. The coordinates of the head in question will be printed in the error message.

1.14.8 Function 7 - RAD Head Analysis for RAD Two

This function is similar to Function six.

1.14.9 Function 8 - RAD Head Analysis for RAD Three

This function is similar to Function six.

1.14.10 Function 9 - RAD Head Analysis for RAD Four

This function is similar to Function six.

1.14.11 Function 10 - RAD Exerciser

This function will exercise a specific section of any of the four RADs on the E channel, or all the RAD units as a composite. There are eight Function Variables:

CORLO	The lowest usable core address is 27000.
CORHI	Variable to 177700.
RADLO	Variable from 00000.
RADHI	Variable to 77777.
FIXBLK	A specific block length in sectors can be forced.
CYCLES	The function will repeat according to the cycle count. If CYCLES is = -1, the function will exit only if forced by a CONTROL input.
PATTERN	Any octal configuration can be used as a pattern.
MODE	The MODE word defines a set of changeable operation types.

Early is defined as the transmission of two separate blocks of data with minimum latency or wait. Not early is defined as random delay or latency between transmissions.

The configuration of one bits in the MODE word determines the type of operation as follows:

Rad Addresses (Bits 0-2)	001-Random 010-Sequential 100-Fixed
Core Addresses (Bits 3-5)	001-Random 010-Sequential 100-Fixed

Data (Bits 6-8)	001-Random 010-Sequential 100-Fixed
Transmission Mode (Bits 9-11)	000-Random SKS, early or interrupt 001-Same as 000 010-Random, early, without interrupt 011-Random, early with interrupt 100-Fixed, not early, without interrupt 101-Fixed, not early, with interrupt 110-Fixed, early, without interrupt 111-Fixed, early, with interrupt

The RAD drive for either buffer one (Bits 12-14) or buffer two (Bits 15-17) is determined as follows:

000-buffer not selected
001
010-Random selection of either
011 read or write
100-buffer not selected
101-Fixed read only
110-Fixed write only
111-Will result in read only

For further details on MODE word usage see Unit 12, Function 10 abstract.

Bit 18 is set for Data chaining mode.

Bit 21 is set to print all errors. If it is a zero, only the first three errors are printed. However, the error count is kept for the entire sector and will be displayed when the sector is fully checked.

1.14.12 Function 11 - Partial Sector Test

This function tests the inhibited increment and zero pad on partial sector blocks. There are three variables:

RAD ADRS	This defines which sector.
WD COUNT	Defines the word count less than one sector or 77 (octal) words.
PATTERN	Any octal configuration can be used.

1.14.13 Function 12 - Read RAD Utility

This function can be used to read any sector on the RAD. The print utility can be used to display the sector data. There is one variable RAD ADRS. This determines which sector is to be read.

1.15 UNIT 13 F CHANNEL RAD

Not installed.

1.16 UNIT 14

Not assigned.

1.17 UNIT 15 W CHANNEL AND 9367 RAD TEST

Unit 15 includes a failure analysis for the W channel, but in all other respects it is identical to Section 1.14.

1.18 UNIT 16 Y CHANNEL AND 9367 RAD TEST

Not installed.

1.19 UNIT 17

Not assigned.

1.20 UNIT 18 E CHANNEL DISC DIAGNOSTICS

Not installed.

1.21 UNIT 19 F CHANNEL DISC DIAGNOSTICS

Not installed.

1.22 UNIT 20

Not Assigned.

1.23 UNIT 21 - W CHANNEL DISC DIAGNOSTICS

This unit tests the disc on channel W.

There are three unit parameters:

FAW	Function activation word, one bits contained in this word correspond to functions that are to be activated in the automatic mode. Legal functions are functions 1, 2, 3, 4, 5, and 10. Functions 18 through 23 do not need to be selected, since they are special and are not activated in the automatic mode.
D00T17	Activation bits for arms 0 through 17 (octal). Bits in the most significant portion of this word permit the corresponding discs to be used. The absence of a bit prohibits a disc from being used. I.e., if disc 10 (octal) is not to be used, the corresponding bit (Bit 8) would be a zero.
D20T37	Activation bits for arms 20 through 37 (octal). This variable combines with the variable D00T17 to provide selective control for all discs. These variables have priority over all function variables, permitting discs to be skipped within a large operating range of discs. I.e., for a 32 Disc System, D00T17, D00T17 = 73777400, D20T37 = 37777400, HIDISC - 777777, and L0DISC = 0 would allow all discs to be used except for discs 3 and 20 (octal). These variables are preset when the unit is loaded according to the system variable "DSCSIZ".

A few conventions must be followed when using this unit.

Changing unit variables – Normally, changing the unit variables requires that a unit transfer (-U 21T) be executed. However, in most cases, changing "D00T17" and "D20T37" does not require restarting the unit, and a continue (T transfer) is sufficient. If the file is keyed in the exerciser with a disc deactivated, an error will result if a read attempt is made on that disc (see function 10).

Changing function variables – Normally when changing function variables, a function transfer is necessary to check and use the change. The special functions are the exceptions to this rule.

Special functions – Functions 18 through 23 are special functions and are not normally accessed. If the operator wishes to use one of them, he must execute a function transfer to that function. When the function is entered, an identification message is printed on the error device and control returned to the keyboard. At this time, the operator should set the function variables to his requirements and type T. When the operation is completed, the ID message will be output and control returned to the keyboard. A new function will not be entered without operator intervention.

Automatic operation — In order to increase the efficiency of the system when running in the automatic mode, all diagnostics will be run and the disc will be keyed in the first pass. When the disc has been keyed, a flag is set in control (it is not a variable), and the unit will be dismissed. When the unit is restarted, only those diagnostics which do not destroy the integrity of the disc will be run (Functions 1, 2, and 5), as well as the exerciser, regardless of the function activation word.

If the disc is software write-protected, functions which destroy the integrity of the disc will be skipped. These are functions 3 and 4. In addition, the exerciser is forced to a special running mode. (See Function 10.)

1.23.1 Function 0

Not assigned.

1.23.2 Function 1 - TMCC Diagnostic

This function tests several basic TMCC operations without the use of an external device. The interlace registers, interrupts, and several skips are tested. If an error is detected, it is reported a source location(s) and logic page(s). I.e., 20D43 (25) indicates that the suspected problem is located on card 20D, Pin 43, and the signal can be found on logic page 25.

There are no function variables.

1.23.3 Function 2 - Controller Diagnostic Without Data Transfer

This function tests as many operations of the disc file controller as possible without involving data transfer. The address register, skips (legal and illegal), header verification, and position verification are checked.

Because of the complexity of the disc file controller, an attempt is made to give as much information as possible when an error occurs. Information displayed usually includes a brief indication of the failure, logic equations that are directly involved, source modules, and logic pages.

The following assumptions are made:

Write header switch is off.

Write protect switches are all up or not write protected.

Error/stop switch is in the continue mode.

File is on-line.

Headers are good.

The object tests which use discs deleted from "D00T17" and "D20T371" will be skipped.

There are no function variables.

1.23.4 Function 3 - Controller Diagnostic with Data Transfer

This function tests disc file controller operations while transferring data, parity generation and checking, address registration incrementing, termination of various states are included. Error reporting is the same as function 2.

Object tests using discs deleted from "D00T17" and "D20T37" will be skipped. In addition, if the disc is software write-protected or has been previously keyed, this function will be skipped.

There are no function variables.

1.23.5 Function 4 - Header Verification and Addressing Test

This function verifies the ability to address the entire disc file and also verifies all headers. During the first pass, the first word of each sector is tagged with its own address. In the second pass, the first word of the sector is checked. If the data does not compare with the address, an error message will be printed. If an I/O error is detected, an appropriate error message will be printed. No attempt is made to diagnose the error.

Discs which have been deleted from "D00T17" and "D20T37" will be skipped. If the disc has been previously keyed or is software write-protected, this function will be skipped.

There are no function variables.

1.23.6 Function 5 - Data Products 5045 Disc File Diagnostic

This function contains object tests which are designed to locate some of the problems in the data products 5045 disc file. Some of the position decoder (PDBA) logic, the timing, and the clear logic is tested.

Object tests using discs deleted from "D00T17" and "D20T37" will be skipped.

There are no function variables.

1.23.11 Function 10 - Disc Exerciser

This function exercises the disc in one of several different modes, automatically or under operator control. The eight function variables are as follows:

OPMODE	Control word for mode of operation
LOCORE	Starting core address. This must be greater than 34000

HICORE	Ending core address. For a 925/930, maximum core address is 37777. For a 940, it is 177777.
LODISC	Starting disc address
HIDISC	Ending disc address. Maximum disc is 777777 for a 32 disc system.
LENGTH	Control for transmission length. If LENGTH is negative random length records will be used. If LENGTH is positive it is the fixed record length, in sectors, to be used. For a 940, the maximum fixed length is 340 sectors (14K); if the OPMODE is set for Compare Mode, the maximum fixed length is 160 sectors (7K). The length cannot be greater than the difference between HICORE and LOCORE.
PATTERN	Data that will be transmitted in the fixed data mode.
COUNTERS	Bits 9 through 12 are the number of retry attempts to be made if an I/O error is detected. Bits 18 through 23 are the number of data errors to be displayed after the first error detected in a given sector.

The bits in the variable OPMODE have the following significance:

0 - Fixed disc addressing (uses address in LODISC)

1 - Sequential disc addressing

2 - Random disc addressing

3 - Fixed core addressing (uses address in LOCORE)

4 - Sequential core addressing

5 - Random core addressing

6 - Fixed data (pattern is used as a data word. The sector address is added to last word in the sector).

7 - Sequential data (disc address is in most significant 18 bits).

8 - Random data

9 - Not used

10	-	Compute while transferring data
11	-	Use interrupts
12	-	Buffer 1 operation fixed (read or write)
13	-	Write Buffer 1
14	-	Read Buffer 1
15	-	Buffer 2 operation fixed (read or write)
16	-	Write Buffer 2
17	-	Read Buffer 2
18	-	Not used
19	-	Compare mode
20	-	Key mode
21	-	Execute dummy seek before each disc access
22	-	Time all seeks
23	-	Time all searches

In the compare mode, operation is controlled by the status of Buffer 1. If Buffer 1 is fixed read, a read-read-compare-write-read-compare operation will result. This operation will not destroy the integrity of the disc. If Buffer 1 is fixed write, a write-read-compare operation will result to allow data to be checked as it is being written.

In the key mode, the disc will be keyed with the selected data. The unit will not be dismissed, until the keying is complete.

There are several parameter combinations which are not allowed in the exerciser. These will be flagged as errors.

Disc addressing not specified.

Core addressing not specified.

Data not specified.

Fixed core, fast mode.

Buffer 1 fixed operation, write and read.

Buffer 2 fixed operation, write and read.

No Buffer selected.

Compare mode, Buffer 1 operation not fixed.

Compare mode, Buffer 2 operation not fixed.

Compare mode, Buffer 2 not read.

Compare mode, length random.

Fixed length too large.

Fixed length greater than 14K (340B sectors)

Compare mode, length greater than 7K (160B sectors).

Fixed length = 0

Locore less than 34000B.

Hicore greater than 177777B.

Hicore greater than 37777B, not 940.

Hicore minus locore less than 64D (1 sector).

Hidisc less than lodisc.

Hidisc greater than 777777.

Key mode, disc addressing not sequential, buffer 1 or 2 fixed in read mode.

IO STATUS	an indication of the state at the time of failure.
ERR FLAG	a flag used with IO status to indicate which error was detected.
TIS-TSB	time is and time should be for positioning time errors.
SRT DISC	starting disc address.
END DISC	ending (pinned) disc address.
SRT CORE	starting core address.

END CORE	ending (pinned) core address.
BLK LENGTH	transmit block length.

Bits in the word IO status and err flag have the following significance:

0	-	File not on line
1	-	Controller not ready (500 ms timeout)
2	-	Controller error set
3	-	Track not verified
4	-	Disc write protected (during write attempt)
5	-	Write header switch on
6	-	Not used
7	-	Seek time error
8	-	Not used
9	-	Search time error
10	-	Not used
11	-	Not used
12	-	Channel error set
13	-	Channel active (500 ms timeout)
14	-	Word count not zero
15	-	Not used
16	-	Not used
17	-	Not used
18-20	-	Current retry number
21-23	-	Current phase

0	-	Inactive
1	-	Disc seek

2 - Disc seek (retry)

3 - Write Buffer 1

4 - Read Buffer 1

5 - Write Buffer 2

6 - Read Buffer 2

If a data error is detected, the following information is displayed:

WORD IS	Bad data
WORD SB	Good data
DISC ADD	Disc address of bad data
CORE ADD	Core address of bad data
START DISC	Starting disc address
LENGTH	Transmitted block length
WORD NO	Word number within the sector
ERROR NO	Error number within the sector

In the compare mode, WORD IS comes from Buffer 2 and WORD SB comes from Buffer 1.

When function 10 is entered, if the disc has not been keyed, the key mode is set (OPMODE = 22126610). If the disc has been keyed, the automatic running mode is set (OPMODE = 11133307). If the disc is software write protected, the R-R-C-W-R-C mode is set and will not destroy the integrity of the disc (OPMODE = 11135527) see compare mode explanation.

In order to reset the sequential disc pointer, type O 15271T. This pointer is not reset by typing F 10T. The pointer will be reset and control returned to the keyboard. If T is typed, a function 10 transfer will be executed.

1.23.19 Function 18 - Write Protect Switch Test

This special function tests the status of the write protect switches by positioning the arms to position 0 and then testing the switches. If a disc is found to be write-protected, the message 'write protected - disc XX' will be printed on the error device.

There are two function variables.

START	Starting arm number 0-37 (octal)
END	Ending arm number 0-37 (octal)

The function will continue to run until BP4 is toggled. Discs deleted from the variables 'D00T17' and 'D20T37' will be skipped.

1.23.20 Function 19 - Single Increment vs. Time Plotter

This special function times the arm movements from position 0 to position 1 to position 2 etc., ending at position 63. At this point, the motion is reversed and the time is measured from position 63 to position 62 to position 61, etc., until position 0 is reached. The times for all movements are then entered into a graph which is output on the error device. The symbols used are:

Plus Sign	Forward direction times
Minus Sign	Reverse direction times
Delta Sign	Equal forward and reverse times

Discs will be sequentially tested using the 'start' variables. If a disc does not come ready within 500 milliseconds, an error message is output and the test aborted.

The graph is scaled as follows:

Horizontal Scale	Ending arm position
Vertical Scale	Positioning time in milliseconds

Typical range in times is from 140 ms to about 200 ms. There are no existing specifications to indicate just what are good and bad times, however, in many cases, a problem has been indicated by irregularities in the graphs. For example, a disc file that is rotating too slowly, and a disc file with timing logic set incorrectly both had graphs whose minimum positioning times were 5 to 10 milliseconds too long (effectively displacing the graph). Other problems such as sticky arms will result in high positioning times. In several cases, comparing the graph of a suspected bad arm, with that of a good arm has shown problems.

1.23.21 Function 20 - Multiple Increment vs. Time Plotter

This special function times all possible combinations of 1, 2, 3, . . . 64 positions moved and records the maximum and minimum times for each increment value on a graph. Discs are tested sequentially using the START and END variables. For each arm is approximately 18 minutes. Then run time graph is output to the line printer.

The graph is scaled as follows:

Horizontal Scale	-	Number of positions moved.
Vertical Scale	-	Maximum and minimum positioning times in milliseconds.

Typical positioning times range from 140 to 350 milliseconds depending upon the amount of positions moved.

If the disc does not come ready within 500 milliseconds, an error message is output and the disc is aborted.

1.23.22 Function 21 - Write Header

This special function will write the headers on sequential addresses according to the variables START and END. These variables are in the form of disc pot words. START should have an address starting at sector 0, head pair 0. I.e., 777600 is disc 37, track 77, header pair 0, sector 0.

1.23.23 Function 22 - Write Header Test

The purpose of this function is to provide the operator with a tool for use in locating problems encountered in header writing. No attempt is made to diagnose errors, just to provide a program for use while scoping. BP1 must be set.

The function variables are START and END, which are the starting and ending disc addresses in the form of disc pot words. The same variables will be used until breakpoint 1 is reset, at which point the control will revert to the keyboard. The sector count must range from 1 to 128.

1.23.24 Function 23 - Sector Dump

The purpose of this special function is to provide the operator with a means of dumping one sector on the disc to the error device. The control will be returned to the keyboard when the print is completed.

1.24 UNIT 22 Y CHANNEL DISC DIAGNOSTICS

Not Installed.

There is one variable:

SECTOR	Address to be read and printed.

1.25 UNIT 23 CTE - 10/11 TEST

This unit test is composed of 16 functions designed to test the CTE-10 and one CTE-11 unit.

To use this unit test, the user must insert the CTE-11 loop test card into slot 3 of those CTE-11 Units he wishes to test.

Function 1	will test channels 0-3
Function 2	will test channels 4-7, etc.
Function 16	will test channels 74-77

The channels to be tested must be set as bits in the system variable SYSIZE. Each bit corresponds to four channels or one CTE-11.

Bit Position	Channel
0	0-3
1	4-7
2	10-13
3	14-17
4	20-23
5	24-27
6	30-33
7	34-37
8	40-43
9	44-47
10	50-53
11	54-57
12	60-63

Bit Position	Channel
13	64-67
14	70-73
15	74-77

The program assumes the following standard cable connections. Cable W303 in slot 1 of the CTE-10 will address channels 0-17, where channels 0-3 will be on the first CTE-11, channels 4-7 will be on the second CTE-11, channels 10-13 will be on the third CTE-11 and channels 14-17 will be on the fourth CTE-11.

Cable W304 in slot 2 of the CTE-10 will address channels 20-37, where channels 20-23 will be on the first, channels 24-27 on the second, channels 30-33 on the third and channels 34-37 will be on the fourth CTE-11.

Cable W305 in slot 3 of the CTE-10 will address channels 40-57, where channels 40-43 will be on the first, channels 44-47 on the second, channels 50-53 on the third and channels 54-57 on the fourth CTE-11.

Cable W305 in slot 4 of the CTE-10 will address channels 60-77, where channels 60-63 will be on the first, channels 64-67 on the second, channels 70-73 on the third and channels 74-77 on the fourth CTE-11.

The following tests are performed for every channel:

Test the POT and PIN connections to verify that the CTE-10 is addressable.

Test the scanner to verify that it is changing values.

Test the ready SKS under channel activated and deactivated conditions.

Test the channel status SKS under channel activated and deactivated conditions.

Acknowledge and verify the ON and OFF interrupts.

Test the channel addressing capability, verify that the POT and PIN word has identical addresses when acknowledging an interrupt.

Test the transmit buffer empty SKS when the buffer is empty.

Acknowledge and verify the receive interrupt and test the transmit buffer not empty SKS.

Acknowledge and verify the transmit buffer empty interrupt.

Transmit a character consisting of all ones to the channel.

Acknowledge the receive interrupt and verify the character.

Test the over-run bit and verify that it is not set by a single character transmission.

The error messages are designed to give the following information:

A brief description of the detected error.

A message directing the user to the specific CTE-11 unit.

Signals and module locations which can cause the error. All CTE-10 module locations are denoted by AXX whereas all CTE-11 module locations are denoted by JXX.

The user can find the specific channel the error was detected on by executing the following on the keyboard.

OI

If the user looks at the listing at the address AAAAA he will find the specific channel causing the error.