Offline Diagnostics System Manual

HP 3000 Series 900 Computers
HP 9000 Series 800 Computers

HP Part No. 30190-90010
Printed in USA March 1991

Edition 6 - E0391
FOR HP INTERNAL USE ONLY

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

System Technology Division
19483 Pruneridge Avenue
Cupertino, CA 95014

Printing History

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

The software code printed alongside the date indicates the version level of the software product at the time the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual updates.

First Edition	August 1987
Second Edition	December 1987
Third Edition	May 1988
Fourth Edition	December 1988
Update 1	March 1989
Fifth Edition	December 1989
Update 1	October 1990
Sixth Edition	March 1991

List of Effective Pages

The List of Effective Pages gives the date of the current edition and of any pages changed in updates to that edition. Within the manual, any page changed since the last edition is indicated by printing the date the changes were made on the bottom of the page. No information is incorporated into a reprinting unless it appears as a prior update.

All	December 1987
All	December 1988
Title Page	March 1989
2	March 1989
3	March 1989
4	March 1989
3-1 to 3-41	March 1989
4-1 to 4-40	March 1989
All	December 1989
1-1 to 1-5	October 1990
4-1 to 4-49	October 1990
5-3 to 5-4	October 1990
9-1 to 9-76	October 1990

Safety and Regulatory Information

For your protection this product has been tested to various national and international regulations and standards. The scope of this regulatory testing includes electrical/mechanical safety, radio frequency interference, ergonomics, acoustics, and hazardous materials. Where required, approvals obtained from third-party test agencies are shown on the product label. In addition, various regulatory bodies require some of the information under the following headings.

USA Radio Frequency Interference

The United States Federal Communications Commission (in 47CFR Subpart J, of Part 15) has specified that the following notice be brought to the attention of the users of this product:

Warning

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested for compliance with the limits for Class A computing devices pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

Japanese Radio Frequency Interference

The following notice is for users of this product in Japan:

この装置は，第一種情報装置(商工業地域において使用されるべき情報装置)で商工業地域での電波障害防止を目的とした情報処理装置等電波障害自主規制協議会（VCCI）基準に適合しております。

従って，住宅地域またはその隣接した地域で使用すると，ラジオ，テレビジョン受信機等に受信障害を与えることがあります。

取扱説明書に従って正しい取り扱いをして下さい。

Japanese Radio Frequency Notice

Safety Considerations

This product and related documentation must be reviewed for familiarization with safety markings and instructions before operation. The following figure shows some of the safety symbols used on the product to indicate various safety considerations.

SAFETY SYMBOLS

Instruction manual symbol: the product will be marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect the product against damage.

Indicates hazardous voltages.

Indicates earth (ground) terminal (sometimes used in manual to indicate circuit common connected to grounded chassis).

Warning

The WARNING sign denotes a hazard. It calls attention to a procedure, practice, or the like, which, if not done correctly or adhered to, could result in injury. Do not proceed beyond a WARNING sign until the indicated conditions are fully understood and met.

Caution

The CAUTION sign denotes a hazard. It calls attention to an operating procedure, practice, or the like, which, if not done correctly or adhered to, could damage or destroy part or all of the product. Do not proceed beyond a CAUTION sign until the indicated conditions are fully understood and met.

Preface

This manual contains information about the Offline Diagnostics System for the HP 3000 Series 900 and the HP 9000 Series 800 Precision Architecture RISC Computer Systems. It is intended to be used as technical support documentation for Hewlett-Packard CEs, CEC Engineers, SEs, and other qualified support personnel. The procedures and software focus primarily on the hardware troubleshooting environment, and require specific training for correct and safe usage.

Specifically, this manual describes the Offline Diagnostics System, as well as its programs and utilities. It is used as an operation and reference manual, once the system has been loaded and launched via the Support Tape media.

Contents

Tables

Contents

1

Overview

Introduction

The Offline Diagnostics System provides a means of testing System Processor Unit (SPU) hardware Field Replaceable Units (FRUs) and interrogating low-level hardware register contents. It includes a standard operating environment complete with a library of common procedures, program macros, and command set/feature functionality. The Offline Diagnostics and Utilities are implemented via the Support Tape. This tape is available in either open reel or cartridge format.

The purpose of the Offline Diagnostics and Utilities is to provide intelligent computer-based troubleshooting support in the absence of a functional operating system (MPE XL or HP-UX). The capabilities of the Offline Diagnostics System support two kinds of users. They are:

- Field Support Personnel (HP & OEM) who are troubleshooting SPU failures. These users will generally use the "command-message" interface described in the Chapter 2.
- Expert Support Personnel engaged in low-level hardware debugging activity. These users are more likely to use the low-level "binary" command interface. More expert levels of use also require detailed knowledge of both system internals (Initial System Loader & Operating System) and Processor-I/O Dependent Hardware code for any given SPU under investigation.

All chapters provide operating instructions, detailed test/function/command descriptions and example sessions. Error message information is incorporated into test descriptions or included at the end of the chapter. Major changes for the current edition are noted at the beginning of each chapter.

The available diagnostic tests are:

Name	Systems	Description
A1002AP	825/925/835/935/845/949/922/832/932	A1002A SPU Proc. Diag
A1100AP	850/950/855/955/960/870/980	A1100A SPU Proc. Diag
A1002AM	825/925/835/935/845/949	A1002A SPU Memory Diag
A1100AM	850/950/855/955/960/870/980	A1100A SPU Memory Diag
A1002AI	825/925/835/935/845/949/922/832/932	A1002A SPU I/O Diag
A1100AI	850/950/855/955/960/870/980	A1100A SPU I/O Diag
IOMAP	all PA-RISC systems	Input/Output Map Utility
CAEXR	all PA-RISC systems	Channel Exerciser Utility
BCDIAG	825/835	Bus Converter Diagnostic
UNIPROC	870/980	PCX Uniprocessor Diagnostic
MPROC	870/980	Multiprocessor Diagnostic

Note

The utility, SADPATCH, is also available. For further information on SADPATCH, see the *MPE XL SE Utilities Reference Manual, P/N 32650-60035*.

All Offline diagnostics are loaded and called by ISL. They conform to the ISL subset of the Object Module specification. This subset removes portions of the Object Module header, thereby reducing the code space requirements.

Offline Diagnostics System Requirements

In order to support the Offline Diagnostics System, an HP Precision Architecture computer must be able to boot the Initial System Load (ISL) environment and provide all of its functionality. User access to at least one functioning terminal is also required. The following events and functions must occur to meet these requirements:

- Processor Dependent Code (PDC) must successfully execute.
- PDC must boot and transfer control to the ISL environment.
- Offline Diagnostic program code must be available in Logical Interface Format (LIF) files.

Boot Files - Logical Interchange Format (LIF)

System boot files are stored on tape or disc as Logical Interchange Format (LIF) files. The purpose of the LIF scheme is to ensure and facilitate file transfer and compatability among all HP computer systems. It is a Hewlett Packard standard for file construction on any bootable media. The standardization of file format for the pre-operating system boot environment guarantees that one ISL based Offline Diagnostics System environment will be loadable on all HP Precision Architecture RISC computer systems.

The LIF files are stored on the lowest blocks of the boot media, in the order listed below.

LIF File Volume Label

LIF file to the Processor Dependent Code (PDC).

Non-LIF File Label

loading and initializing either the HP-UX or MPE XL operating systems.

LIF File Directory

load utilities.

Initial Program Load (IPL) code

program software. This program provides a primitive pre-operating system environment that allows specification and manipulation of the Primary boot, Alternate boot, and Console paths. It permits system boot options such as steering the SPU into either an ISL Offline based (Single User) utility/program or an MPE-XL/HP-UX operating system (Multiple User) environment. In MPE XL systems, an additional program, MMSAVE, stores a core image prior to loading ISL.

One or more LIF Data/Program files

located and loaded by the ISL program as selected by the user or a LIF directory autoboot file resident in non-volatile memory.

Loading the ISL Environment

PDC implements all processor dependent functions; such as initialization and selftest. After successfully completing selftest, PDC retrieves the LIF File Directory and begins the bootstrap process. In order to load ISL, PDC must know the particular device on which ISL resides. Typically, the device is occupies the primary boot path that is maintained by PDC.

HP UX Systems

Load (IPL) file. PDC then loads this program, which boots the system to the ISL level. At the ISL level, the boot process can be broken to run the Offline diagnostics.

MPE XL Systems

PDC then loads MMSAVE, which writes an image of the pre-existing memory to disc as a troubleshooting aid. After this is completed, PDC loads IPL, which boots the system to the ISL level. At the ISL level, the boot process can be broken to run the Offline diagnostics.

If ISL Autoboot is enabled then PDC gives a ten second delay during which time the operator may override the autoboot sequence by entering any character on the console. If the autoboot sequence is overridden or not enabled in the first place, PDC prompts the operator for the boot path to use. Any required path components that are not supplied default to zero. The Primary boot, Alternate boot, Console paths, and autoboot enable may be modified via ISL commands. Boot path selection procedures are described fully in Chapter 2.

Purpose of ISL Environment

The Initial System Loader (ISL) implements the operating system independent portion of the bootstrap process. It provides an intermediate stopping point in the boot process where the operator can modify the boot path or run an Offline diagnostic.

All PA-RISC SPUs contain special purpose memory for maintaining critical configuration related parameters (e.g. Primary boot, Alternate boot, and Console paths). Two forms of memory are supported: Stable Storage and Non-Volatile Memory (Non-Volatile Memory is not present on SPUs tested by the A1002A SPU Processor Diagnostic).

Typically, when control is transferred to ISL, an autoboot sequence takes place. Typically, ISL executes commands from the autoexec file in a script-like fashion. During autoboot, ISL displays its revision and the name of any utility it executes. However, when autoboot is disabled the user will be prompted for input after ISL displays its revision. Acceptable input is any ISL command name or the name of any utility available in the LIF directory. If a non-fatal error occurs or the executed utility returns, ISL again prompts the user for input.

Offline Diagnostics System Components

The Offline Diagnostics System is composed of the User Interface (UI) and diagnostic programs. Because they run from the ISL environment rather than MPE XL or HP UX, the system is unavailable for normal use. The user interface (UI) operates as an integral part of the three diagnostic programs, SPU Processor (A1002AP/A1100AP), SPU Memory (A1002AM/A1100AM), and SPU I/O (A1002AI/A1100AI). This user interface cannot be accessed directly by the user but functions automatically whenever any of the above programs are invoked. The Offline utility programs IOMAP and CAEXR contain their own user interface.

User Interface

The User Interface (UI) is the communication link between the user and the various diagnostic programs. It sends messages to the user from diagnostic programs, and returning user replies. Because the Offline Diagnostics System interface provides two levels of access, Command-Message and Binary. These are described in Chapter 2.

Diagnostic Programs

The Diagnostic Programs are a comprehensive set of software to test FRUs for Processor, Memory and I/O functionality on PA-RISC SPUs. These diagnostics determine which of the field replaceable units (FRUs) need replacement.

Utility Programs

Offline Utility programs cannot isolate defective FRUs, but can verify which functions of a device are operating correctly. Input/Output Map (IOMAP) and Channel Exerciser (CAEXR) help determine the cause of device failure by providing stress simulation and diagnostic information.

These utilities can provide identification/loopback verification of the internal I/O modules and in some cases invoke Selftests for I/O and peripheral devices. System exercisers provide a means of using or maximally loading a particular part of the system. The Channel Exerciser program provides a way of using I/O hardware under stress conditions that equal or exceed those expected under maximum load.

Both of these utilities have their own user interface which is both interactive and command driven.

Program Construction

Each diagnostic consists of sections and steps. Sections are the smallest portion of a diagnostic which can be repeated and expected to return "no errors" on known good hardware. A step generally tests a specific circuit for one or more particular failures. The steps in a section are generally grouped by the hardware tested.

Support Tape Implementation

The ISL-based Offline Diagnostics and Utilities are implemented via the Support Tape in open reel, cartridge tape, or DDS format.

Defect and Enhancement Requests

Submit defect reports and enhancement requests through the STARS database. Refer to the appropriate product numbers, listed below.

Product Number	Test Name	Product Name
30342-10001	A1002AP	SPU Processor Diagnostic
30342-10002	A1002AM	SPU Memory Diagnostic
30342-10003	A1002AI	SPU I/O Diagnostic
30343-10001	A1100AI	SPU I/O Diagnostic
30343-10002	A1100AM	SPU Memory Diagnostic
30343-10003	A1100AP	SPU Processor Diagnostic
30344-10001	IOMAP	Input/Output Map Utility
30345-10002	CAEXR	Channel Exerciser Utility
30344-10001	ISL	Initial System Loader
30345-10003	BCDIAG	Bus Converter Diagnostic
30343-10005	UNIPROC	PCX Uniprocessor Diagnostic
30343-10006	MPROC	Multiprocessor Diagnostic

Contents

2

User Interface (A1002A/A1100A SPU Diagnostics)

Introduction

The User Interface (UI) provides a standard execution environment for Offline SPU diagnostics. Numerous commands and parameters are available to start, stop, and monitor the progress of diagnostic programs. A Help facility provides the user with information on any of the UI commands. The UI environment is used primarily by support personnel as a command-message interface for troubleshooting purposes.

More expert users can utilize this interface from the binary perspective for register examinatiou with the debug command (LDB) and the DUMP, STATE, or REGISTER commands. The debug facility provides methods to examine or modify hardware register contents or modify a diagnostic unavailable through the UI.

Comparison with Online Diagnostics Interface

The Online Subsystem is designed to execute in an operating system environment. By contrast, the Offline system was designed to execute under a very simple executive, ISL. Consequently, features relatiug to multi-tasking, file system, protection, security, synchronization, etc., are not found in the Offline system. However, the Offline system has features which allow a greater degree of user control over the diagnostic program code.

The UI of the Offline Diagnostics System resembles the DUI of the Online Diagnostics Subsystem. Offline UI commands and their similarity to Online DUI commands are described below.

Run Command

The run command exists on both the Offline and Online systems but has a different meaning in the offline environment. Here, it prints a banner message and waits for input. In the online environment, it executes the default diagnostic.

Resume Command

If the program has been suspended, "resume" restarts it at the point of suspension.

Note

The function of the "run" and "resume" commands varies between SPU Processor, Memory and I/O diagnostics. For more information, see "Console Commands and State Variables" later in this chapter.

Shared Commands

The following command exists in both the Online and Offline diagnostic systems:

DEBUG	EXIT
HELP	LOOP
SECTION	ERRONLY
ERRPAUSE	

Note

The command "HARDCOPY" appears in the banner command list but is currently unsupported.

Offline-Only Commands

CHANGIO	Enable/Suppress Error Pause (EEPS/SEPS)
LISTIO	Enable/Suppress Error Print (EEPR/SEPR)
RESET	Enable/Suppress Non-Error Pause (ENPS/SNPS)
STATE	Enable/Suppress Non-Error Print (ENPR/SNPR)
STOP	Enable/Suppress Isolation Pause (EIPS/SIPS)
DUMP	Enable/Suppress Isolation Print (EIPR/SIPR)
REGISTER	Displays register values

Unique Commands

Check the "help" and "info" screens for more information. These commands are described under the subsection "Unique Commands" in each diagnostic chapter.

Command - Message Interface

The diagnostic system contains a core set of command input and message output functions. The standardized messages, message types, pauses, command grammar, and standardized commands are explained in this section.

Command Entry Rules

The diagnostic system console I/O is synchronous; neither the outputs nor the inputs are pipelined. The user cannot enter a new command while an existing command is being processed. A command may not be entered while output is being displayed to the terminal. The user cannot queue output to the terminal.

Console Output

The diagnostics have three kinds of message outputs to the system console, error messages, isolation messages, and nonerror messages.

Error Messages

These communicate specific deviations from expected results. Error messages exist only for this purpose. Error messages consist of two parts, a standard output and a specific output.

The standard output of an error message describes where in the diagnostic the error was detected. The location is given in terms of the section number and step number. The section

and step numbers describe the error condition in a broad sense and then explain it in more detail.

The actual output of an error message provides more detail about the specific hardware deviation than is given by the section and step numbers. The output may be as long as is required to communicate the detail required by the user. An example is shown below:

```
ERROR IN SECTION 001, STEP 012.                (standard output)
GENERAL REGISTER 1 WAS 6, SHOULD HAVE BEEN 4.  (specific output)
```

Isolation Messages

Isolation messages communicate the action that the user should take as a result of a detected error or describe the action undertaken by the diagnostic as a result of an error. Consequently, there are two forms: terminal and branch. The terminal form recommends what should be done to fix a hardware failure. It is a character string of unlimited length. A branch notifies the user of a transfer of execution to another section of the diagnostic which can further isolate the error. At that point, the diagnostic generates a terminal isolation message.

An example of an error message followed by terminal isolation message is:

```
ERROR IN SECTION 001, STEP 012.
GENERAL REGISTER 1 WAS 6, SHOULD HAVE BEEN 4.
REPLACE XYZ CHIP ON ABC BOARD.
```

An example of an error message followed by a branch isolation message is:

```
ERROR IN SECTION 001, STEP 012.
TIMEOUT ERROR ON READ FROM REGISTER 8.
ISOLATION BRANCH TAKEN.      (This message is a non-error print.)
SECTION 004
ERROR IN SECTION 004, STEP 040.
TIMEOUT ON WRITE TO REGISTER 2.     (This message is an error print.)
REPLACE XYZZY CHIP ON WEG BOARD.
```

Non-Error Messages

Messages which are neither error message or isolation messages are non-error messages. Non-error messages can be grouped into two classes, prompts and terminated output .

Prompts

Prompts are short messages which precede certain requests for input. There are two kinds of prompts, console command prompts and diagnostic programmer prompts. Console command prompts are always four characters long (three letters followed by a ">"). There is no termination character. Each diagnostic has only one prompt.

Prompts are generally automatic, although the user can control some prompts following non-error messages.

Diagnostic programmer prompts may be a variable number of characters long. There is no termination character. Diagnostic commands prompt the operator for configuration information, environmental information, and so forth.

Terminated Output

There are several standard, terminated non-error messages: banners, section activity indicators, step activity indicators, and illegal command responses. These messages cannot be suppressed by the user.

Banner Message

Contains the diagnostic name and version. It may also include additional information. Banners always include the diagnostic system supplied messages:

```
Beginning of SOM address = xxxxxxxxx
Beginning of code space address = xxxxxxxxx
Last code space address = xxxxxxxxx
Last SOM address = xxxxxxxxx
DIAGNOSTIC SYSTEM FOR ISL STANDALONE DIAGNOSTICS
Version x.x.

Type HELP for command information. (This is a non-error message print.)

Type INFO for test information.
```

Section Activity Indicator

is defined to be the message "SECTION NN", where "NN" represents the section number in decimal. Diagnostics contain up to 128 sections. Refer to the Console Command descriptions for more detail.

Step Activity Indicator

defined to be the the step number, in decimal format. For example, a section four with fifteen steps could print the following activity indicators after the completion of the first four steps:

```
SECTION 004
029 030 031 032
```

The same section would print the following indicators by the end of the section:

```
SECTION 004
029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044
```

Activity indicators do not prompt after message output.

Illegal Command Response

The diagnostic system detects an illegal command, the system sends an "ILLEGAL COMMAND" message followed by the first four characters of the erroneous command. (Certain ASCII control characters are converted iuto blanks.)

Pauses

After a diagnostic sends an output message, it can either continue or it can prompt the user for input. After an error or isolation message, the diagnostic may prompt the user for a diagnostic system command. After a non-error message, the diagnostic may prompt the user for a diagnostic system command or prompt the user for data, but not both at the same time.

At the end of every diagnostic there is a non-error message which states "DIAGNOSTIC COMPLETED.". The pause after this message can not be suppressed by the user.

Command Syntax

The command input syntax is based on alphanumeric characters in the decimal and hexadecimal numbering systems. All lower case letters are converted to their upper case equivalents by the diagnostic system on input.

A delimiter is any of the following characters: space ( ), semicolon (;), carriage return (Return), or null ( ). A space represents the end of the previous word, number, hexadecimal number, or term. Following the space begins the next word, number, hexadecimal number, or term. A ";" means end of the previous command and following the ";" begins the next command. A (Return) means end of record. A null means end of file.

Two adjacent delimiters are interpreted as shown:

Delimiter one	+	Delimiter two	=	Result
space		space		space
space		;		;
space		carriage return		carriage return
space		null		null
;		space		;
;		;		;
;		carriage return		carriage return
;		null		null
carriage return		space		carriage return
carriage return		;		carriage return
carriage return		carriage return		carriage return
carriage return		null		carriage return
null		space		null
null		;		null
null		carriage return		null
null		null		null

The diagnostic system interprets characters as follows:

- Characters following either a carriage return or a null are ignored.
- The first character(s) on a command input line must be either a letter or spaces. Leading spaces are removed by the diagnostic system until a letter is found. If the first character in a command input line is a letter, then it is assumed to be part of a command word.
- A special character is any printable character other than letters or decimal digits or hexadecimal digits.

Input Definitions

Definitions of input types are defined below.

word	A variable length string of letters terminated by a delimiter and preceded by a delimiter or first character.
decimal number	A variable length string of decimal digits preceded and terminated by a delimiter. Negative numbers are not allowed as command input.
hexadecimal number	A variable length string of hexadecimal digits preceded and terminated by a delimiter. Only logical integers are allowed as command input. (X'FFFFFFFF -> D'4,295,000,000) The distinction between a word and a hexadecimal number in cases of syntactical ambiguity is command specific according to the following rule: the first string of characters in a command is always interpreted as a word intended to be a command.
term	A variable length string consisting of letters, decimal digits, hexadecimal digits, or special characters preceded and terminated by a delimiter.
parameter	{a word | a decimal number | a hexadecimal number | a term}.
parameter list	Abbreviated "pl", is { a parameter | [parameter] repeated a variable number of times}.
command string	{a word | a word, a pl}
command line	{ a command string terminated by a cr| [a command string terminated by a ";"] repeated a variable number of times, a command string terminated by a Return)}

Syntax/Grammar Examples

The grammatical rules listed above are illustrated in the following examples:

- These character sequences are words: `Z, xz, abz, abcdefgHIJKlmnopqrstuvwz`
- These character sequences are not words: `a1, B+, %abcd`
- These character sequences may be either words or hexadecimal numbers: `A, AB, BAD, ACED, FFFFFFFF`
- These character sequences are terms: `!AD, +, ^^, {([/|\])}, -9`
- The following character sequence illustrates two equally valid command statements: "`do good;add 4 + 4`(Return)"
- The following lines illustrate how successive terms are nested within a command input statement. They are all equivalent commands:

```
"     fix board  ;  ;;;  ;; add   4  + 6;;;; (Return)"
"     FIX BOARD  ;  ;;;  ;; ADD   4  + 6;;;; (Return)"
" FIX BOARD ; ; ; ADD 4 + 6; (Return)"
"FIX BOARD;ADD 4 + 6(Return)"
```

  The variations in case and spacing do not affect the meaning of the command. The additional semicolons (;) do not affect the relationship between the three major elements: FIX BOARD, ADD, (Return).

  The character sequence "AAAA AAAA(Return)" is the "AAAA" command followed by the word or hexadecimal character string "AAAA". The code implementing the "AAAA" command determines the significance of the "AAAA" parameter.

  When the diagnostic system detects a syntax error it prints: SYNTAX ERROR.

Console Commands & State Variables

The diagnostic system command structure consists of four modes: diagnostic execution control, diagnostic output control, diagnostic system control, and debug.

Diagnostic Execution Control Commands

These commands control which portion of a diagnostic is executed or the number of loops.

SECTION {integer: 0 ... 127 | integer integer | integer/integer | [integer] [integer/integer] }
This command sets a 128 bit logical integer bit mask. Each bit corresponds to a test section in the diagnostic. If the bit is set, the test section is execnted. If a bit is set for a section which does not exist, then the diagnostic experiences a small performance delay and continues. SECTION 4 sets bit four and clears all other bits. SECTION 4 6 sets bits four and six and clears all other bits. "integer/integer" is an increasing bit range. SECTION 4/6 is equivalent to SECTION 4 5 6. SECTION 6/4 is illegal.

LOOP [32-bit logical integer] This command sets a thirty-two bit logical integer counter. Before the execution of each diagnostic pass, the counter is compared with zero. If it is zero, then the diagnostic returns to the diagnostic command interface. The diagnostic may range limit or range check the counter. If the diagnostic range checks the counter it must use one of the standard error reporting schemes if the counter is out of range. If no count integer is supplied with the command, then the counter will not be decremented and the diagnostic program loops infinitely. "LOOP 1" means loop once. "LOOP 0" means do not loop at all; this is legal syntax and can be used to turn off looping for the next resume command. "LOOP" usually means loop forever since the count is never decremented.

RUN This command unconditionally forces the diagnostic system to branch to the point in the diagnostic where the banner is printed. This is the first command to issue to start executing the diagnostic. The diagnostic may contain code after this point which is executed as a result of the command c001c186R. This command cannot cause a stack overflow. It always executes in the same manner if the hardware has not changed.

PROCESSOR DIAG: displays banner and halts.

MEMORY DIAG: reloads diagnostic, displays banner and begins diagnostic.

I/O DIAG: reloads diagnostic, displays banner and I/O information, then halts.

RESUME This command resumes normal diagnostic execution after a pause. It also resumes diagnostic execution after a pause or suspension.

PROCESSOR DIAG: execute diagnostic, restart when suspended.

MEMORY DIAG: (A1002A), begin diagnostic at banner. (A1100A), restart diagnostic at banner.

I/O DIAG: At beginning of execution, prints banner, I/O table, then configuration table, execute diagnostic. Within diagnostic, restarts diagnostic from banner.

RESET This command resets the following diagnostic state variables to an initial condition suitable for field use. It is set to a value determined by the diagnostic programmer. The "Hardcopy" flag is also cleared.

EEPS	Enables error pause. It enables the diagnostic system to pause after the detection of an error and accept any diagnostic command.
SEPS	Suppresses the pauses after errors.
ENPS	Enables a non-error pause. It enables the diagnostic system to pause and accept any diagnostic command.
SNPS	Suppresses non-error pauses.
EIPS	Enables isolation pause. The command enables the diagnostic system to pause after the detection of an isolation and accept any diagnostic command.
SIPS	Suppresses the pauses after isolations.
ERRPAUSE	A shorthand way of typing EEPS;SIPS;SNPS.

Diagnostic Output Control Commands

These commands enable or disable the printing of certain output messages.

EEPR	Enables error print. It enables the diagnostic system to output error messages.
SEPR	Suppresses the output of error messages.
ENPR	Enables non-error print. It enables the diagnostic system to output non-error messages.
SNPR	Suppresses the output of non-error messages.
EIPR	Enables isolation print. It enables the diagnostic system to output isolation messages.
SIPR	Suppresses the output of isolation messages.
ERRONLY	A shorthand way of typing EEPR;SIPR;SNPR.
HARDCOPY	(Not supported at this time.) Causes all messages to the console to be echoed to a second terminal connection for communication with a printer.

Diagnostic System Control Commands

These commands permit examining diagnostic system state or changing diagnostic system state variables.

STATE — This command displays the state variables for the diagnostic system which appear as shown below:


```
PRG>STATE

                STATE VARIABLES

                ---------------
            (E=ENABLED  S=SUPPRESSED)

Error Print       E   Non-Error Print   E   Isolation Print  E
Error Pause       E   Non-Error Pause   E   Isolation Pause  E
Infinite Looping S    Hardcopy          S

Section           N       Step                N
Total loops       N       Loops remaining     N
Step Address      Return

Selected Sections
0-31     Mask0        Section 32-63       Mask1
64-95      "          Section 96-127        "
```

An "E" or "S" means a Boolean value. An "N" means an integer. A Boolean value is equal to either "E", enabled or "S", suppressed. "Section", "Step", "Loop", and "Count" are represented as decimal logical integers. "Mask0", "Mask1", and "Return" are hexadecimal integers.

"Loop" is both an integer and a boolean value because looping can be disabled.

The "Section" and "Step" variables communicate the actual test section and step being executed. "Loops remaining" is the number of loop passes remaining during the current execution.

"Mask0" and "Mask1" are thirty-two bit integers which represent the 128 bit section mask. This eight digit hexadecimal number is decoded from left to right with the leftmost number representing the LEAST significant value of the section number displayed. For example "40000000" in mask 0 means that section 1 has been selected, and "7E000000" in this same mask indicates that sections 1 through 6 have been selected.

"Return" lists the return address the diagnostic goes to if the user enters the "RESUME" command.

STOP — Upon exiting the diagnostic, this command stops ISL from continuing the Auto-Boot sequence.

LISTIO — Lists the I/O path to the primary and secondary boot devices, the console path, and the printer path. The output appears as follows:

```
PRG>LISTIO

BOOT 1     :AA.BB.CC.DD.EE.FF.GG
BOOT 2     :AA.BB.CC.DD.EE.FF.GG
CONSOLE    :AA.BB.CC.DD.EE.FF.GG
PRINTER    :AA.BB.CC.DD.EE.FF.GG
UUT 1      :AA.BB.CC.DD.EE.FF.GG
UUT 2      :AA.BB.CC.DD.EE.FF.GG
```

(UUT means Unit Under Test)

```
or if a device is not configured:

PRINTER:-- DEVICE PATH NOT CONFIGURED --
UUT 1  :-- DEVICE PATH NOT CONFIGURED --
UUT 2  :-- DEVICE PATH NOT CONFIGURED --


This output is consistent with ISL.
```

CHANGEIO — {path name} [C,P,1,2] This command changes the console (C), the printer (P), and the UUT's (1 and 2) displayed by the LISTIO command. The diagnostic system has a volatile memory buffer in which the six I/O paths are stored. This memory buffer is initialized by the diagnostic system by either reading values from stable storage or supplying default constants from the diagnostic system. In either event, this command only changes the memory buffer. To permanently change the boot and console values found in stable storage, the user must return to ISL.

HELP [keyword] — Prints information regarding the diagnostic system command syntax and semantics. If no keyword is provided then the list of commands is output to the terminal. If a keyword is provided then the information listed for that command is output to the console. Diagnostic specific commands are also keywords.

INFORMATION — Prints a diagnostic specific description of key test information. This command primarily supplies application information, the principal hardware tested by each section, the total number of sections, the default sections, etc. (This function is always diagnostic specific. The diagnostic programmer must customize this function for each diagnostic.)

EXIT — Unconditionally returns the system to the ISL interface.

Control-Y — This character breaks the execution of the diagnostic between steps. It is similar to the Control-Y function available under MPE, the Break function available under RTE, and the Control-X (or -Y) function found in HP-UX. (The A1002A also uses Control-C). Any console command is valid at this point. The I/O associated with console commands is not suppressible.

Notes

1. If a valid command is erroneously invoked with extra parameters, then the error message "TOO MANY ARGUMENTS" is displayed.
2. If the parameters are of the wrong class, then an error message stating "SYNTAX ERROR" is printed.
3. If a command is erroneously invoked with too few parameters then the error message "TOO FEW ARGUMENTS", or "NO ARGUMENTS ERROR!" is printed.

Abbreviations

The diagnostic system supports a form of abbreviation; only the first four characters of a command are significant. Thus, the following two sets of commands have the same meaning:

Command	Abbrev	Command	Abbrev
SECTION (S)	SECT	LOOP (LOOPCOUNT)	LOOP
RUN	RUN	RESUME	RESU
RESET	RESE	EEPS	EEPS
SEPS	SEPS	EIPS	EIPS
SIPS	SIPS	ENPS	ENPS
SNPS	SNPS	ERRPAUSE	ERRP
EEPR	EEPR	SEPR	SEPR
EIPR	EIPR	SIPR	SIPR
ENPR	ENPR	SNPR	SNPR
ERRONLY	ERRO	HARDCOPY	HARD
STATE	STAT	STOP	STOP
LISTIO	LIST	CHANGEIO	CHAN
HELP	HELP	INFORMATION	INFO
REGISTER	REGI	DUMP	DUMP
DEBUG	DEBU	EXIT	EXIT

Scripts

The diagnostic system supports scripting as a simple form of batch access. ISL allows run strings to be passed to the programs it loads and executes. It also has an autoboot feature. These two features are complemented in the diagnostic system by a run string parse facility. The diagnostic system always looks for its next command in the run string passed to it by ISL unless an error is encountered or the run string is exhausted. On error, when the run string is exhausted, the diagnostic system queries the console for input. Together, the ISL and Diagnostic System features allow creation of diagnostics which can execute successfully unattended.

When a diagnostic is executing commands which originate from the ISL run string, the console prompt is ISL_CMD> instead of the usual diagnostic prompt.

Examples

These examples illustrate the relationship between commands and error reporting modes. In these examples, user input always appears following and to the right of either the system, diagnostic, or ISL prompt. Comments concerning the output statements appear to the right in parentheses and italics.

Example 1. Help at the ISL> Prompt

When the ISL prompt is present, enter "help" to display the diagnostic programs. Enter the acronym displayed to load the desired program.

```
ISL> help

    ?             Help Facility
    HELP          Help Facility
    LISTF         List ISL utilities
    LS            List ISL utilities
    AUTOBOOT      Set or clear autoboot flag in stable storage
    AUTOSEARCH    Set or clear autosearch flag in stable storage
    PRIMPATH      Modify primary boot path in stable storage
    ALTPATH       Modify alternate boot path in stable storage
    CONSPATH      Modify system console path in stable storage
    DISPLAY       Display boot and console paths in stable storage
    LSAUTOFL      List contents of autoboot file
    LISTAUTOFL    List contents of autoboot file
    READNVM       Displays contents of one word of NVM
    READSS        Displays contents of one word of stable storage

    Utilities on this system are:

    HPUX
    IOMAP
    CAEXR
    A1002AI
    A1002AM
    A1002AP
    A1100AI
    A1100AM
    A1100AP

ISL>
```

Example 2. Load a Diagnostic

```
ISL> a1100ap

Beginning of SOM address = xxxxxxxxx
Beginning of code space address = xxxxxxxxx
Last code space address = xxxxxxxxx
Last SOM address = xxxxxxxxx
DIAGNOSTIC SYSTEM FOR ISL STANDALONE DIAGNOSTICS
Version x.x.

Type HELP for command information. (This is a non-error message print.)

Type INFO for test information.

Series A1100AP System Processor Unit Diagnostic, Version 1.8
There are 126 Sections in the Diagnostic - 1 thru 126
```

Example 3. Erroneous Command

This example shows three errors detected by the diagnostic system. PRG> is the prompt of an unspecified diagnostic program:

```
PRG> fumblinG fIngers
ILLEGAL COMMAND = FUMB          (Notice the case change.)
PRG> SECTION
NO ARGUMENTS ERROR! command = SECT
PRG> SECTION !4
ARGUMENT ERROR: COMMAND = SECT !4
PRG>                                (New Prompt.)
```

Example 4. Error Example

The following error message shows program response to an incorrect test section specification. In this case, the diagnostic had only 127 sections.

```
PROC>section 1/200

ARGUMENT ERROR! command = SECT 1/200 exceeds section 127.
```

Example 5. Normal Execution: Backplane Problem.

This example demonstrates the print and pause suppress features:

- The program will run the test on an infinite loop (loop parameter)
- The program will run section 4 only (section parameter)
- The "resume" command actually starts the program.
- Pausing and error output are suppressed to speed the program (snpr, seps, sipr, sips parameters)

```
PRG>Loop;SECTION 4
                                    Loop on section 4 until error or control-Y.)
PRG> RESUME
                                                        (Execution begins.)
SECTION 004
050 051 052 053 054 055 056 057
END OF PASS 0001
SECTION 004
050 051 052 053
ERROR IN SECTION 004, STEP 053.
Value is 0001. Should be 0003. (The error message can be lower case.)
REPLACE EITHER PROCESSOR, BACKPLANE, OR I/O FRU's
PRG>snpr;seps;sepr;sipr;sips
                              (Set the diagnostic up for oscilloscope use.)
>RESUME
                         (The manufacturing technician tests the)
                         (backplane for termination problems.  Notice that)
                         (the diagnostic now runs at mazimum speed.)
                         (The technician finds the bad resistor.)
[Control]  [Y]           (Technician intercepts diagnostic.)
PRG>EXIT
```

Binary (Debug) Interface

Note Low Level Debug (LDB) is not currently supported and may not be available on all systems.

As stated in the Command-Message section, the diagnostic system provides access to the PDC debugger, LDB. In general, the specific code and data structures visible with LDB are diagnostic specific. However, three instances exist where standardized code and/or data structures are visible and important to the user.

1. The ISL/diagnostic interface is standardized by the Bootstrap, IPL, and ISL Standard.
2. Invocation of the Monitor in PDC is standardized.
3. The diagnostic system supplies an interrupt handler which is standardized across diagnostics, unless diagnostic specific code modifies or replaces the interrupt handler.

ISL Structure

The interface between ISL and the diagnostic system has three essential points.

1. A diagnostic is a System Object Module, (SOM). This SOM, however, has some special restrictions: it has no external references, and it has a special auxiliary header which omits much of the information found in the standard header.

   ISL performs all the address range checks required to assure that a diagnostic can be safely loaded. ISL insures that the diagnostic does not overlay itself and that the loaded program will fit in physical memory. ISL does not prevent one loaded program from overlaying another program.

2. ISL invokes a diagnostic by treating it as a space local procedure call. The specific procedure call variables are defined as follows:

```
GR2       ISL return address

GR25      Address of Boot Info String
GR26      Pointer to PDC entry point
GR28      Flag register
GR29      (Possible) Pointer to next ISL Command String
GR30      Stack Pointer
```

3. It is permissible for a diagnostic to use the same stack used by ISL provided the stack size requirements do not exceed 7Kb. GR28 and GR29 only contain return values for ISL.
4. The interface between ISL and the diagnostic system is a special variation of the PA-RISC Procedure Calling convention.
5. ISL always performs an initialization of the console and bootpath upon return from a diagnostic (or anything else). It makes use of the ENTRY_INIT entry point of IODC.

LDB Load Activity

The diagnostic system provides a procedural link to the monitor contained in PDC. The link code performs the following visible external functions when calling LDB:

- GR2, GR31, and CR31 are saved on the stack in this order. These values are the old return pointer, the old link register, and the old temporary register 7. LDB uses CR31 to implement its break trap.
- GR31 is loaded with the entry address of LDB. GR2 is loaded with the return address. This address points to code within the Debug procedure. Upon return from LDB to the Debug procedure, GR2, GR31, and CR31 are restored to their previous values and these values are deleted from the stack. Finally, the Debug procedure returns to its caller.

Except for GR2, GR31, CR31 and PC, all other register values are unchanged.

Debug Commands

These commands allow the user to examine a diagnostic data structure in detail, modify a diagnostic, or debug a diagnostic.

DEBUG

This command invokes the Software Maintenance Monitor, LDB. (LDB will be available in a future release.) It is possible to return to the diagnostic system. If the machine on which the diagnostic system is executing does not support the PDC entry point for LDB, then the error message "PDC Debugger is not supported" is be printed. (PDC debugger is not supported on the HP A1002A SPU.)

DUMP

As in DUMP{INTEGER | INTEGER/INTEGER | [INTEGER] [INTEGER/INTEGER] } This command displays a range of real memory addresses. All parameters are 32 bit logical addresses. Space register 4 is presumed to be used even though real addresses are generated. The INTEGER syntax is a contraction of the INTEGER/INTEGER syntax where the two integers are equal. All addresses are rounded down to the nearest word boundary. If only one word is in the address range, then exactly one word is read and printed. If more than one word is in the address range, then multiple words are read and displayed. Each line of the display always displays four words, even if this exceeds the requested range.

Caution

This command reads the requested address or address range even if this action would cause an HPMC.

This command exists to circumvent some of the limitations of SPU's that lack LDB. The format of the display is:

```
xxADDRESS: AAAAAAAA


or for a range of integers

xxADDRESS: AAAAAAAA BBBBBBBB CCCCCCCC DDDDDDDD
yyADDRESS: EEEEEEEE FFFFFFFF GGGGGGGG HHHHHHHH
```

REGISTER

This command displays the general registers that were last sampled with the SAMPLE_GRs procedure described within the programmatic perspective. The saved registers are displayed according to the following format:

```
PRG>REGISTER
 ------------------------------------------------------------------------------
 General Registers : sampled within : Section xxx, Step yyy
 ------------------------------------------------------------------------------
 r00 / AAAAAAAA BBBBBBBB CCCCCCCC DDDDDDDD EEEEEEEE FFFFFFFF GGGGGGGG HHHHHHHH
 r08 / IIIIIIII JJJJJJJJ KKKKKKKK LLLLLLLL MMMMMMMM NNNNNNNN OOOOOOOO PPPPPPPP
 r16 / QQQQQQQQ RRRRRRRR SSSSSSSS TTTTTTTT UUUUUUUU VVVVVVVV WWWWWWWW XXXXXXXX
 r24 / YYYYYYYY ZZZZZZZZ aaaaaaaa bbbbbbbb cccccccc dddddddd eeeeeeee ffffffff
 ------------------------------------------------------------------------------
```

Interrupts

The diagnostic system provides two default interrupt handlers; one for Powerfail warning and one for all others. An exception to this is the processor diagnostic, which has its own trap handlers and messages. The default response for every interrupt except powerfail detection is to capture the architected processor state and to optionally print that state with an error message:

```
UNEXPECTED INTERRUPT! Interrupt number is XX.
{the text equivalent of the interrupt number goes here}
Processor State Trace Follows

General Registers in numerical order
gr0: AAAAAAAA BBBBBBBB CCCCCCCC DDDDDDDD EEEEEEEE FFFFFFFF GGGGGGGG HHHHHHHH
r08: AAAAAAAA BBBBBBBB CCCCCCCC DDDDDDDD EEEEEEEE FFFFFFFF GGGGGGGG HHHHHHHH
r16: AAAAAAAA BBBBBBBB CCCCCCCC DDDDDDDD EEEEEEEE FFFFFFFF GGGGGGGG HHHHHHHH
r24: AAAAAAAA BBBBBBBB CCCCCCCC DDDDDDDD EEEEEEEE FFFFFFFF GGGGGGGG HHHHHHHH
Control Registers in functional order
cr17/18 (pcq): AAAAAAAA AAAAAAAA BBBBBBBB BBBBBBBB cr20/21: CCCCCCCC CCCCCCC
EIRR: DDDDDDDD RC: EEEEEEEE IT: FFFFFFFF IIR: GGGGGGGG IPSW: HHHHHHHH
EIEM: AAAAAAAA IVA: BBBBBBBB CCR: CCCCCCCC SAR: DDDDDDDD
PIDS: EEEEEEEE FFFFFFFF GGGGGGGG HHHHHHHH
c24: AAAAAAAA BBBBBBBB CCCCCCCC DDDDDDDD EEEEEEEE FFFFFFFF GGGGGGGG HHHHHHHH
```

"XX" is the Interrupt Vector Address (IVA) index associated with the interrupt and is a decimal number. The IVA register is significant to the diagnostic system but does not affect ISL, which does not use interrupts.

The diagnostic system powerfail warning response consists of three steps. First, the processor data cache is flushed in an architected manner. Second, PDC_POW_FAIL is called. If PDC_POW_FAIL returns, then the message

```
Power was not lost.
```

is printed and the diagnostic returns to the interrupted code.

If AC power is lost but battery power is preserved until AC power is restored, then the diagnostic system attempts a powerfail recovery by doing an implied RUN command. The diagnostic system restores the value of the IVA upon termination of a diagnostic.

If for some reason the PON signal is noisy, the SPU repeatedly and intermittently executes PDC selftest. Likewise, a TOC should also cause the SPU to execute selftest and reboot ISL. The diagnostic system prints an error message if the powerfail vector, word 4, or the TOC vector, word 8, are not zero upon entry into the diagnostic.

Example

This example may not execute with the current release.

```
>Debug          (when available, the LDB display will appear.)

Registers:  General
```

```
 r0  / 00000000 00000000 00002580 00000000  00000000 00000000 00000000 00000000
 r8  / 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
 r16 / 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
 r24 / 00000000 00000000 00000000 F1F1F1F1  F1FFF1F1 00001020 00001040 00000000
 pc  = 00000000.00000000   priv = 0   psw = 00000000    sar = 0  (decimal)
Program (Real Mode)
```

```
   >  00000000              / E8001222   B N             00000918
      00000004              / E8020580   B               000042CC
      00000008              / 08000240   OR              r0 r0 r0
      0000000C              / 20602000   LDIL            L'0x1000 r3
      00000010              / EAA10EEC   BL              000037A0 r21
      00000014              / B4630002   ADDI            0x1 r3 r3
   1  00000018              / 8C0031BA   COMIBF = N      0 r2 00000910
      0000001C              / 23C00000   LDIL            L'0x0 r30
      00000020              / B41E07FF   ADDI            -0x1 r0 r30
   Data (Real Mode)
```

```
00000000/E8001222 E8020580 08000240 20602000 EAA10EEC B4630002 8C0031BA 23C0000
00000020/B41E07FF 8b004120 21602302 B43A0133 E10321B5 08000240 31000205 B901440
Command
```

```
   LDB>
         (The GR2 will point to the return address of the debug procedure.)
   LDB>quit
   PRG>                    (Return to the diagnostic command interpreter.)
   n
```

Contents

Tables

3

SPU Processor Diagnostic (A1002AP, A1100AP)

Introduction

The Series HP A1002A/A1100A SPU Processor Diagnostic tests the VLSI chip set of the Central Processing Unit (CPU) for FRU failures. The VLSI chip set includes a Central Processing Unit (CPU), Translation Lookaside Control Unit (TCU), one or more Cache Controller Units (CCU's), a System Interface Unit (SIU) and a Floating Point Controller (FPC). The FPC contains two math chips: ALU and Mult/Div.

Systems Tested by the A1002AP SPU Processor Diagnostic

The following table lists all current system families tested by the A1002AP SPU Processor Diagnostic.

Table 3-1. Systems Tested by A1002AP SPU Processor Diagnostic

Family	Label	Code Name	Models
xx2	xx2	Silverfox	922, 832, 932, etc.
825/925	x25	Firefox	825, 925
	x35	Topgun	835, 935, etc.
	x45	Shogun	845, 949, etc.

Systems Tested by the A1100AP SPU Processor Diagnostic

The following table lists all current system families tested by the A1100AP SPU Processor Diagnostic.

Table 3-2. Systems Tested by A1100AP SPU Processor Diagnostic

Family	Label	Code Name	Models
850/950	x50	Cheetah	850S, 950, etc.
	x55	Jaguar	855S, 955, etc.
	x60	Cougar	960
	x80, x70	Panther	870, 980

Note

The code names listed in the two preceding tables are not official; however you may see them in internal documents or hear them used by factory personnel.

Chapter Organization

This chapter provides the following information about the HP A1002A/A1100A SPU Processor Diagnostic:

- Functional Description
- Test Section Descriptions
- Hardware/Test Section Mapping
- Example Session: A1002AP SPU Processor Diagnostic
- Example Session: A1100AP SPU Processor Diagnostic
- Error Message Information

Defects and Enhancements

Submit defect reports and enhancement requests for this diagnostic through the STARS database, referencing number 30342-10001 (A1002AP) and 30343-10003 (A1100AP).

Minimum Configuration

The minimum hardware/software configuration required to load and run the offline diagnostics system consists of the following functional hardware, firmware, and software items:

- HP Precision Architecture RISC SPU
- Boot device and boot path hardware (such as a magtape drive)
- System console
- Initial System Load Code (ISL)
- Offline Diagnostics Software

Functional Description

The user is strongly urged to carefully read and understand this information before attempting to use this diagnostic for the first time. Online help is available to the user with the ISL based Offline Diagnostics Environment "Help" facility.

The diagnostic program provides a structure to selectively perform the following functions: execute sections, print activity messages for sections and steps, print error and isolation messages, and pause after an error or isolation message. When looping is enabled, all selected sections are executed during each pass through the loop.

The diagnostic strategy checks the hardware functionality by verifying that all data/instruction/control pathways conform to the HP Precision Architecture RISC specifications. These specifications are standard for all PA-RISC SPUs. Any hardware that fails to respond as expected implies that it has a hard fault and is called out as a failed FRU. The SPU Processor diagnostic is grouped into eight sets of tests as follows:

Table 3-1. SPU Processor Diagnostic Test Sets

Set	Name of Test	Sections	Tests
1	CPU Data Path	1/6	6
2	SIU Data Path	7/10	4
3	CCU0 Data Path	11/18	8
4	CCU1 Data Path	19/26	8
5	TCU Data Path	27/40	14
6	CPU Instruction	41/93	53
7	CPU Extended	94/102	9
8	Floating Point	103/126	24

Because test sets three and four are identical, there are seven actual sets of processor tests. The sets are explained below. The specific details of each test section are explained in the section, "Test Section Descriptions".

Test Set	Description
1	**CPU Data Path** Checks the functionality of the data path hardware in the CPU chip. The six sections exercise registers, the adder, shifter and associated logic functions. Each test section aborts as soon as an error is detected; an error number is loaded into the error register (gr28) and control is returned to the sequencer.
2	**SIU Data Path** Tests the data path hardware in the SIU chip. The four sections test all accessible registers, the timer (cr16), the interrupt mechanism, and most SMB transactions. Some of these sections contain conditional assembly statements which modify the tests for usage on all SPUs tested by the A1002AP SPU Processor Diagnostic.
3 & 4	**CCU Data Path** Tests the data paths and general functionality of each CCU and its associated cache RAMs. The eight sections execute twice (once for each CCU). The test enables the replacement counter in one cache bank and disables the replacement counter in the other. All code execution takes place from the cache band with the replacement counter enabled, while actual test data is performed ou the other. After each test section is completed, both banks are set back to their normal state, and the process is repeated for the next section.
5	**TCU Data Path** Consists of 14 sections which test the data paths and general functionality of the TCU and its associated TLB RAMs. An additional file (tcuutl) contains several utility routines called by the other TCU test sections.
6	**CPU Instruction** Consists of 53 sections which test the iustruction set. These sections verify adherence to the PA-RISC computer architecture, and account for implementation variants in the PA-RISC SPU.
7	**CPU Extended** Consists of 9 test sections which perform extended tests on CPU functionality by executing groups of instructions, or types of operations, using all possible combinations. Tests cover arithmetic, logical, shifts, load and stores, branches, traps, privileged operations, protection traps, and unit operations.
8	**Floating Point** Consists of 24 test sections which test the data paths and instructions performed by the math coprocessor. Functions such as single and double precision addition, subtraction, multiplication, and division, along with all coprocessor control/data pathways, are checked.

Unique Commands

Three additional commands are available to determine diagnostic information:

PSTAT	Displays the chip revision number and cache line lockout status.
CREGISTER	Displays the contents of the control registers as of the end of the previously executed test section.
PROC n	Selects 1 to 4 processors to test (A1100AP only).

Test Section Descriptions

The processor diagnostic consists of 126 test sections and a control program. The control program manages execution order and interfacing of common procedures provided by the user interface (UI). The test section listing which follows describes the diagnostic sections and the functions tested.

Section	Description
1	Tests the general registers by writing four patterns to them: 1's, 0's, alternate 0's, and alternate 1's, and then checking that they contain the proper values. It also checks for crosstalk between registers and data retention by writing the register number into each register, and checking for the correct number after waiting for period of time (4k clocks).
2	Tests the CPU ALU by doing adds, subtracts, logical operations and shift and add instructions with alternating 0's and 1's patterns.
3	Tests the CPU shifter by shifting 2 different patterns by 1,2,4,8,16 and 31 bits and checking that the correct results are obtained. It also checks the shift amount register, c11, with alternating 0's and 1's.
4	Tests conditions which are possible with arithmetic unit, extract/deposit, and carry/borrow instructions.
5	Tests the 8 carry/borrow bits in the PSW by doing decimal arithmetic using the DCOR instruction and checking that the correct results are obtained. A second set of tests uses the ADDC and SUBB to check the carry/borrow.

6	Tests the following control registers located on the CPU chip:
	`cr0 recovery counter`
	`cr10 coprocessor configuration register`
	`cr14 interrupt vector address register (iva)`
	`cr18 pc offset queue (pcoq)`
	`cr19 interrupt instruction register (iir)`
	`cr22 interrupt process status word (ipsw)`

7	Pattern-tests the temporary registers (tr0 thru tr7, aka cr24 thru cr31), the External Interrupt register (EIR), the Interval Timer Limit (ITL1M), Interval Timer Counter (ITCNT), the Timeout Counter (TMOUT) and the FLEX and STAT registers.

Section	Description
8	Tests the functionality of the ITCNT, checks the chip rev#, tests access to SMB I/O registers, tests that timer match will set bit 0 of EIR and directed writes to EIR will set all other bits, and verifies that moves to cr23 will reset EIR bits. It also checks that bits set into EIR will cause external interrupts, attempts to reset the SIU will cause a Higher Priority Machine Check (HPMC), and tests the functionality of the Read Line, shared and Purge Data Cache operations, and tests that TMOUT resets and increments on SMB read operations.
9	Tests bits in the SIU Status register (STAT). The bits tested include the non-responding module (NRM), the time-out bit (TMO), the Force Parity Error bit (FOR), the Slave Parity bit (SLP), and the Other Module Error bit (OME).
10	Tests the READ HALFLINE and CLEAR, and the COPYOUT SMB transactions.
11	PN7F: CCU0 Diagnose commands and registers. PN10: ICCU Diagnose commands and registers.
12	PN7F: CCU0 RAM Array pattern tests. PN10: ICCU RAM Array pattern tests for Set 1.
13	PN7F: CCU0 Dirty bit setting PN10: ICCU RAM Array pattern tests for Set 2
14	PN7F: CCU0 data and tag parity checker functionality PN10: ICCU cache addressing
15	PN7F: CCU0 Valid bit setting and dirty bit resetting PN10: ICCU parity and hamming generation and error detection
16	PN7F: CCU0 Purge Data Cache instruction and valid bit resetting. PN10: ICCU RPN registers and comparators.
17	PN7F: CCU0 Flush Data Cache instruction. PN10: ICCU HPMC and LPMC traps.
18	PN7F: CCU0 Lock functionality. PN10: ICCU Flush functionality.
19	PN7F: CCU1 Diagnose commands and diagnose registers. PN10: DCCU Diagnose commands and registers.
20	PN7F: CCU1 RAM Array pattern tests. PN10: DCCU RAM Array pattern tests.

Section	Description
21	PN7F: CCU1 Dirty bit setting. PN10: DCCU RAM Array pattern tests for Set 2.
22	PN7F: CCU1 data and tag parity checker functionality. PN10: DCCU Cache addressing.
23	PN7F: CCU1 Valid bit setting and Dirty bit resetting. PN10: DCCU Load and Store operations.
24	PN7F: CCU1 Purge Data Cache instruction and valid bit resetting. PN10: DCCU Parity and Hamming generation and error detection, single bit error correction.
25	PN7F: CCU1 Flush Data Cache Instruction. PN10: DCCU RPN registers and comparators.
26	PN7F: CCU1 Lock functionality. PN10: DCCU HPMC and LPMC traps.
27	PN7F: TCU diagnose commands and the diagnose, space, and control registers. PN10: DCCU private bus functionality.
28	PN7F: TLP RAM array (TLB tag parity). PN10: DCCU Flush and Purge functionality.
29	PN7F: RPN RAM array (Data and Instruction RPN field, the RPN parity field, and I/O Device bit). PN10: TCU diagnose commands and the diagnose, space, and control registers.
30	PN7F: TLB RAM Array (Data and Instruction SID fields). PN10: TLB lock bit functionality.
31	PN7F: TLB RAM Array (Data and Instruction VPN fields). PN10: TLB RAM array.
32	PN7F: TLB RAM Array (Data and Instruction PID fields). PN10: Data and Instruction Cache hashing and TLB addressing.
33	PN7F: TLB RAM Array (Data and Instruction ACR fields). PN10: TLB parity generation, purge and insert functionality, read and write access rights checking, access ID checking, PID register functionality, and RPN I/O bit.
34	PN7F: TLB RAM Array (Valid bit and Flag bits). PN10: HPMC traps on TLB and RPN field errors.

Section	Description
35	PN7F: Hashing function. PN10: Parity generation and checking logic.
36	PN7F: Purge TLB instructions. PN10: Data Lookaside Buffer.
37	PN7F: Lock bit functionality. PN10: Instruction Lookaside Buffer.
38	PN7F: Tag and RPN parity checkers. PN10: TLB HPMC traps with all transactions.
39	Tests the functionality of virtual data translations.
40	Tests access rights functionality and further tests the PC space queue.
41	This section begins the instruction tests by performing simple checks of the B (a special case of BL with gr0 as the link register), COMIBT, COMIBF, and ADDI instructions.
42	Tests the B (branch) instruction with a wider range of displacement values than section 41.
43	Tests the Compare and Branch instructions, COMBT and COMBF. Tests are made with all possible condition fields and with various displacement values.
44	Tests the functionality of the Space Register moves, MTSP and MFSP, and the control register moves, MTCTL and MFCTL.
45	Tests the Status Register Modification instructions, MTSM, RSM, and SSM.
46	Tests the functionality of the LDI instruction, which is an assembler construct for LDO.
47	Tests the Branch Vectored (BV) instruction.
48	Tests the Load instructions, LDW, LDWM, and LDBS, and does some limited tests with the ADD instruction.
49	Tests the functionality of the RFI and Insert instructions, IITLBA and IITLBP.
50	Further tests the IITLBA and IITLBP instructions, and also tests the GATE and PITLB instructions.
51	Tests the functionality of the data TLB instructions, IDTLBA, IDTLBP, PDTLB, PROBER, PROBEW, PROBERI, and PROBEWI. It also tests the absolute load and store instructions, LDWAS and STWAS.

Section	Description
52	Tests the Branch instructions, B and BV, using virtual addressing.
53	Tests the variable extract instructions, VEXTRU and VEXTRS.
54	Tests the variable deposit instruction, VDEP.
55	Tests the ADD, ADDIO, ADDC, ADDCO and ADDL instructions.
56	Tests the ADDI, ADDIO, ADDIT and ADDITO instructions.
57	Tests the AND instruction.
58	Tests the Compare and Clear instruction, COMCLR.
59	Tests the OR instruction.
60	Tests the SUB, SUBO, SUBB, SUBBO, SUBT and SUBTO instructions.
61	Tests the Unit Exclusive Or instruction, UXOR.
62	Tests XOR instruction.
63	Tests the Add and Branch instructions, ADDBT, ADDBF, ADDIBT, and ADDIBF, for conditional branching and nullification of the following instruction with all possible condition codes.
64	Tests the Add Immediate Left instruction, ADDIL, for correct results with a variety of immediate values.
65	Tests the And Complement instruction, ANDCM, for correct results, and conditional nullification of the following instruction with all possible condition codes.
66	Tests the Deposit instructions, DEPI, VDEPI, DEP, ZDEP, ZDEPI, ZVDEP, and ZVDEPI.
67	Tests the Extract instructions, EXTRU and EXTRS, for correct extraction, and conditional nullification of the following instruction with all possible condition codes.
68	Tests the Load Space ID instruction, LDSID.
69	Tests the Move and Branch instructions, MOVB and MOVIB.
70	Tests the Subtract Immediate instructions, SUBI and SUBIO.
71	Tests the Unit Add Complement instructions, UADDCM and UADDCMT.

Section	Description
72	Tests for the Taken Branch trap for the BL, BLR, BV, BE, BLE, and GATE instructions. It also tests for an Illegal Instruction trap for a GATE instruction executed in the delay slot of a taken branch.
73	Tests the Compare Immediate and Clear instruction, COMICLR.
74	Tests the Decimal Correct instructions, DCOR and IDCOR.
75	Tests the Shift and Add instructions, SH1ADD, SH1ADDO, SH1ADDL, SH2ADD, SH2ADDO, SH2ADDL, SH3ADD, SH3ADDO, and SH3ADDL.
76	Tests the Shift Double instructions, SHD and VSHD.
77	Tests the absolute address load and store instructions, LDWAX, LDWAS and STWAS.
78	Tests the Data Memory Break trap mechanism, including the PSW X-bit, and the B-bit in the TLB entry.
79	Tests the TLB Dirty Bit Fault trap mechanism.
80	Tests the following instructions for Lower Privilege Transfer, Higher Privilege Transfer, Privileged Operation, and Privileged Register traps when operating at privilege levels lower than the highest. The instructions tested include GATE, RFI, SSM, RSM, LPA, IDTLBA, IDTLBP, IITLBA, IITLBP, PITLB, PDTLB, MTSP, MTCTL, and MFCTL.
81	Tests the Control Register Move instructions, MTCTL and MFCTL, and the control registers.
82	Tests the data memory protection ID validation mechanism.
83	Tests the instruction memory protection ID validation mechanism.
84	Tests the Branch on Bit instructions, BB and BVB.
85	Tests the Divide Step instruction, DS.
86	Tests the Load Offset instruction, LDO.
87	Tests the Load and Store Byte instructions, LDB, LDBS, LDBX, STB and STBS.
88	Tests the Load and Store Word instructions, LDWM, LDWX m, LDWS m, STWM and STWS m.
89	Tests the Load and Store Half instructions, LDH, LDHX, LDHS, STH and STHS.
90	Tests the Store Byte Short instruction, STBYS.

Section	Description
91	Tests the Load and Clear Word instructions, LDCWX and LDCWS.
92	Performs more complete testing on the Branching instructions, BL, BLR, BV, BE, BLE, and GATE, than previous tests.
93	Ensures that instructions that should set the carry/borrow bits in the PSW do set them, and instructions that shouldn't, don't.
94	Consists of 11 parts which perform extended tests on different types of instructions which perform addition and subtraction.
95	Consists of 5 parts which perform extended tests on the four Logical instructions AND, OR, ANDCM, and XOR
96	Consists of 6 parts which test the Shift, Extract and Deposit instructions with varying registers, position and length, and pseudo-random operands.
97	Consists of 6 parts whicb test each load and store instruction with 64 pseudo-random displacements, with each space register.
98	Consists of 7 parts which test various types of branching instructions with pseudo-random word displacements, and for conditional nullification of the following instruction for all possible condition codes.
99	Tests the Overflow and Conditional traps caused by various Add, Shift and Add, and Subtract instructions with the possibility of nullification of the following instruction under conditions generated for the Overflow trap instructions. The Break, Assist Emulation, Illegal Instruction, and Taken Branch traps are also tested.
100	Consists of 4 parts which test the Higher/Lower Privilege Transfer, Privileged Register, and Privileged Operation traps.
101	Consists of 9 parts which test the Protection, Data Memory Break, TLB Miss Fault, TLB Dirty Bit Fault, and Non-access Data TLB Miss Fault traps.
102	Consists of 5 parts, each of which test one of the 5 unit operation instructions. The 1st and 2nd parts test the UXOR and UADDCM instructions. The 3rd and 4th parts test the DCOR and IDCOR instructions. The 5th part tests the DS instruction with pseudo-random operands, for correct results and PSW v-bit settings. Conditional nullification is also tested for all possible condition codes.
103	Tests loads and stores to and from the MIU floating point registers, and the ability to copy from one to another.
104	Tests the ability to make floating point operands positive.
105	Tests the ability of the MIU chip to detect attempted use of reserved registers or operations.

Section	Description
106	Tests the ability of the math coprocessor to do single precision floating point addition.
107	Tests the ability of the math coprocessor to do double precision floating point addition.
108	Tests the ability of the math coprocessor to do single and double precision floating point subtraction.
109	Tests the ability of the math coprocessor to do double precision floating point less-than comparisons.
110	Tests the ability of the math coprocessor to do double precision floating point equals comparisons.
111	Tests the ability of the math coprocessor to do double precision floating point greater-than comparisons.
112	Tests the ability of the math coprocessor to do double precision floating point unordered comparisons.
113	Tests the ability of the ADD/SUB chip to convert single precision floating point operands to double precision.
114	Tests the ability of the ADD/SUB chip to convert double precision floating point operands to single precision.
115	Tests the ability of the ADD/SUB chip to convert single precision floating point operands to single and double precision fixed point format.
116	Tests the ability of the ADD/SUB chip to convert double precision floating point operands to single and double precision fixed point format.
117	Tests the ability of the ADD/SUB chip to convert single precision floating point operands to single and double precision fixed point format with the results truncated.
118	Tests the ability of the ADD/SUB chip to convert double precision floating point operands to single and double precision fixed point format.
119	Tests the ability of the ADD/SUB chip to convert single precision fixed point operands to single and double precision floating point format.
120	Tests the ability of the ADD/SUB chip to convert double precision fixed point operands to single and double precision floating point format.
121	Tests the ability of the MULTIPLY chip to do single and double precision floating point multiplication.
122	Tests the ability of the DIVIDE chip to do single and double precision floating point division.

Section	Description
123	Tests the FPC for single and double precision floating point round.
124	Tests the FPC for single precision instruction queuing and bypassing.
125	Tests the FPC for double precision instruction queuing and bypassing.
126	Tests the FPC for single and double precision floating point square root.

Hardware/Test Section Mapping

The tests verify that all of the data path hardware, all implemented instructions, and all but one of the traps and interrupts (power fail) are tested for hard faults. Table 3-2 indicates the test section which is used to test each functional area within the VLSI chips and the other discrete components on the processor board.

Some functional areas of the SPU cannot be tested directly with this diagnostic. These include the Diagnostic Service Processor (DSP), the clock drivers, the 3-volt regulator, the Revision Port, and the temperature sensor. Malfunction of some of these components prevents the diagnostic from running.

Table 3-2. Processor Diagnostic Organization and Coverage

Chip	Functional Area	Section# Where Tested
CPU	General Registers	1
	Control Registers	6,44,81
	Instr. Pipe/Decode	All
	Data Out Registers	50,51
	Program Counter Pipe	All
	Branch Adder	42,47
	ALU	2,57,59,61,62
	Mask/Merge Shifter	53,54,66,67,74,75
	Result Register	All
	Trap Logic	8,14,55,56,60,72,74, 78/80,82,83,97,98,100
	Instr Decoder	All
TCU	Protection Stack	40
Address Stack		28,30,31,32,33,34
Hashing		35
TLB	Tag RAMs	28,30,31,32,33,34
	RPN RAMs	29
	TTL MUX	29
CCU0	RPN/TAG Stack	11
	Data Stack	11
	Address Stack	14
Cache	Tag RAMs	12
Bank 0	Data RAMs	12

Table 3-2. Processor Diagnostic Organization and Coverage (continued)

Chip	Functional Area	Section# Where Tested
CCU1	RPN/TAG Stack	19
	Data Stack	19
	Address Stack	22
Cache	Tag RAMs	20
Bank 1	Data RAMs	20
SIU	Address Stack	8
	Status Stack	9
	Control Stack	7
	Buffer Stack	9
FPC	FP Registers	103,104
	Status Register	103,104
	Operation/Excep Reg	103,104
	Control Logic	103,104,105
	Add/sub Pipe	106/120
	Multiply Pipe	121
	Divide Pipe	122
	Floating Point Operations	123 - 126

Test Sequence

At the ISL prompt, enter the Processor diagnostic name (A1002AP or A1100AP). The system displays:

```
A1002AP/A1035AP System Processor Unit Diagnostic Version 2.1
There are 126 sections in this diagnostic - 1 through 126
```

This is followed by the diagnostic prompt "PROC>" and the system then waits for user commands. At this point select which processors to test, which sections to run, whether to enable activity printouts, error and isolation messages, error or isolation pauses, or enable looping.

The defaults are:

Parameter	State
test sections	all
activity indicators	enabled
error and isolation messages	enabled
pause after isolation message	enabled
looping or hardcopy	no

To execute the diagnostic type "RESUME". See Chapter 2 "User Interface" for command options and syntax. The first message to appear after the RESUME command is given, assuming processor 1 is enabled, is:

```
DIAGNOSTIC STARTED ON PROCESSOR# 1
```

The test completes in one minute. If activity printouts have been selected, the following display appears as the diagnostic progresses through each section (assuming all sections have been selected):

```
STARTING CPU DATA PATH TESTS - SECTIONS 1/6
001 002 003 004 005 006
STARTING SIU DATA PATH TESTS - SECTIONS 7/10
007 008 009 010
...
```

The following message is displayed after the last test has been executed:

```
DIAGNOSTIC COMPLETED ON PROCESSOR# 1
```

If looping has been enabled, message of the following form is displayed at the end of each pass through the diagnostic:

```
END OF PASS 0000000001
```

If error messages have been enabled and a fault is detected then a message of the following form is displayed:

```
Error no. 003 Detected in Section 012
```

Error numbers range from one to over 100 per section. If isolation messages have been enabled, a message in the following form appears immediately following the error message:

```
THE FAULT IS MOST LIKELY IN THE CPU CHIP
```

In some cases, additional components may be designated if the symptom is such that the fault cannot be limited to only one. The exact function that failed can be determined by referring to the section and error numbers in the diagnostic listing.

Run and Resume Commands

The **Run** command displays the banner and halts, displaying the PROC> prompt.

The **Resume** command executes the diagnostic, observing the current parameter settings.

Example Session: A1002AP SPU Processor Diagnostic

The following text is an example of a diagnostic session using the A1002AP SPU Processor Diagnostic. When the ISL prompt is present, enter the acronym for the desired offline diagnostic program. Enter "help" to display the diagnostic programs.

Several system commands are illustrated below. These are: state, loop, section, resume, and register. In the example, the state command has been used repeatedly to show the before and after conditions present as a result of the diagnostic program running with and without various options. The register command likewise shows the register contents under these different conditions.

Example 1. Call Up the Processor Diagnostic

Call up the A1002AP SPU Processor Diagnostic by entering "A1002AP". For additional information on commands and test sections, enter "help" and "info".

```
ISL> A1002AP

Beginning of SOM address = 0x00021000
Beginning of code space address = 0x00050000
Last code space address = 0x000ECFFC
Last SOM address = 0x000F0C10
DIAGNOSTIC SYSTEM FOR ISL STAND ALONE DIAGNOSTICS
Version 1.0

Type HELP for command information.

Type INFO for test information.

A1002AP.A1035A System Processor Unit Diagnostic, Version 2.0
There are 126 Sections in the Diagnostic - 1 thru 126
PROC>

PROC>help

         Command Keyword Repertoire

 SECTION, LOOP, RESUME, RESET, EEPS, SEPS, EIPS,
 SIPS, ENPS, SNPS, ERRPAUSE, EEPR, SEPR, EIPR,
 SIPR, ENPR, SNPR, ERRONLY, HARDCOPY, STATE,
 STOP, LISTIO, CHANGEIO, HELP, INFORMATION,
 DEBUG, DUMP, REGISTER, RUN, EXIT

 enter Help <keyword> for discrete descriptions.

PROC>info
```

FOR HP INTERNAL USE ONLY

This is the A1002A/A1035A ISL-based Diagnostic program. It is intended to test all programmatically accessible components on the A1002A/A1035A Processor board. The program currently consists of 126 test sections plus two other files which, respectively, provide initialization and control of the diagnostic (podctl), and sequencing of the tests (seq_all). The 126 sections are organized into the following 7 groups:

1. CPU data path tests, Sections 1/6 (6 sections)
2. SIU data path tests, Sections 7/10 (4 sections)
3. CCU0 data path tests, Sections 11/18 (8 sections)
4. CCU1 data path tests, Sections 19/26 (8 sections)
5. TCU data path tests, Sections 27/40 (14 sections)
6. CPU instruction tests, Sections 41/93 (53 sections)
7. CPU extended tests, Sections 94/102 (9 sections)
8. Floating point tests, Sections 103/126 (24 sections).

... *(more information on tests and procedures)*

Example 2. Using the State Command

The next step would be to obtain the current system state. The following dialog shows the state command being used to display the Offline Diagnostics System state variables:

```
PROC> state

                          STATE VARIABLES
                          ---------------
                     (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print E   Isolation Print E
 Error Pause       E   Non-Error Pause S   Isolation Pause E
 Infinite Looping S    Hardcopy         S

 Section         0000000000    Step              0000000125
 Total loops     0000000001    Loops remaining   0000000000
 Step address      0016500C

 Selected Sections
 0-31    7FFFFFFF    32-63  FFFFFFFF
 64-95   FFFFFFFF   96-127  FFFFFFFC
```

Example 3. Effects of Reset Command on State

This example shows that issuing the state command after the reset command displays the post-reset system state.

```
PROC>reset

PROC>state

                          STATE VARIABLES
                          ---------------
                     (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print E   Isolation Print E
 Error Pause       S   Non-Error Pause E   Isolation Pause E
 Infinite Looping S    Hardcopy         S

 Section         0000000000    Step              0000000125
 Total loops     0000000001    Loops remaining   0000000001
 Step address      0010500C

 Selected Sections
 0-31    7FFFFFFF    32-63  FFFFFFFF
 64-95   FFFFFFFF   96-127  FFFFFFFC
```

Example 4. Running and Repeating Diagnostic Sections

The dialog below shows the user setting the following command parameters:

- Run sections 1 through 6 (section parameter)
- Repeat them 4 times (loop parameter)
- Enable error pause (eeps) and erronly options.

The second and succeeding command lines compare the SPU state variables and register contents. This illustrates the syntax for multiple commands on a single line:

```
PROC>section 1/6;loop 4;eeps;erronly

PROC>state;resume

                          STATE VARIABLES
                          ---------------
                     (E=ENABLED   S=SUPPRESSED )

 Error Print      E   Non-Error Print S   Isolation Print S
 Error Pause      E   Non-Error Pause E   Isolation Pause E
 Infinite Looping S   Hardcopy        S

 Section        0000000000    Step             0000000004
 Total loops    0000000004    Loops remaining  0000000004
 Step address     0005820C

 Selected Sections
 0-31    7E000000    32-63  00000000
 64-95   00000000   96-127  00000000

END OF PASS 0000000001

END OF PASS 0000000002

END OF PASS 0000000003

END OF PASS 0000000004
```

```
PROC>register

------------------------------------------------------------------------------
General Registers : sampled within : Section=006, Step=000
------------------------------------------------------------------------------
r00 / 00000000 000EC800 000000AA 00000008  00055D00 FFFFFFF2 00AAAA0A FFFFFFFF
r08 / 08000000 80000000 88888888 FFFFFFFF  7E000000 00000000 00000000 00000000
r16 / 00051200 00000001 00000000 02AAA00A  000ECA80 00000000 00055C1C 00000001
r24 / FFF7E000 FFF7F034 00000006 000EA000  00000000 00100010 0001B830 00000100
------------------------------------------------------------------------------
PROC>state

                          STATE VARIABLES
                          ---------------
                     (E=ENABLED   S=SUPPRESSED )

 Error Print      E   Non-Error Print S   Isolation Print S
 Error Pause      E   Non-Error Pause E   Isolation Pause E
 Infinite Looping S   Hardcopy        S

 Section        0000000000    Step              0000000006
 Total loops    0000000004    Loops remaining   0000000000
 Step address     0005880C

 Selected Sections
 0-31    7E000000    32-63  00000000
 64-95   00000000   96-127  00000000
```

Example 5. Set Section and Loops, Run Diagnostic

This example shows how to:

- Select section 1 to run twice (section1;loop 2)
- Display state variables (state) and register contents (register)
- Re-initialize and run the diagnostic (resume)

```
PROC>section 1;loop 2;state;register


                          STATE VARIABLES
                          ---------------
                    (E=ENABLED   S=SUPPRESSED )

 Error Print      E   Non-Error Print S   Isolation Print S
 Error Pause      E   Non-Error Pause E   Isolation Pause E
 Infinite Looping S   Hardcopy        S

 Section        0000000000    Step              0000000006
 Total loops    0000000002    Loops remaining   0000000002
 Step address     0005880C

 Selected Sections
 0-31    40000000    32-63  00000000
 64-95   00000000   96-127  00000000

--------------------------------------------------------------------------------
General Registers : sampled within : Section=006, Step=000
--------------------------------------------------------------------------------
r00 / 00000000 000EC800 000000AA 00000008  00055D00 FFFFFFF2 00AAAA0A FFFFFFFF
r08 / 08000000 80000000 88888888 FFFFFFFF  7E000000 00000000 00000000 00000000
r16 / 00051200 00000001 00000000 02AAA00A  000ECA80 00000000 00055C1C 00000001
r24 / FFF7E000 FFF7F034 00000006 000EA000  00000000 00100010 0001B830 00000100
--------------------------------------------------------------------------------
PROC>resume;register;state

END OF PASS 0000000001

END OF PASS 0000000002

PROC>Continued ...
--------------------------------------------------------------------------------
General Registers : sampled within : Section=001, Step=000
--------------------------------------------------------------------------------
r00 / 00000000 000EC800 000000AA 00000008  00055D00 FFFFFFF2 00AAAA0A FFFFFFFF
r08 / 08000000 80000000 88888888 FFFFFFFF  7E000000 00000000 00000000 00000000
r16 / 00051200 00000001 00000000 02AAA00A  000ECA80 00000000 00055C1C 00000001
r24 / FFF7E000 FFF7F034 00000006 000EA000  00000000 00100010 0001B830 00000100
--------------------------------------------------------------------------------
```

```
                        STATE VARIABLES
                        ---------------
                    (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print S   Isolation Print S
 Error Pause       E   Non-Error Pause E   Isolation Pause E
 Infinite Looping S    Hardcopy        S

 Section         0000000000    Step              0000000001
 Total loops     0000000002    Loops remaining   0000000000
 Step address      0005700C

 Selected Sections
 0-31     40000000     32-63  00000000
 64-95    00000000    96-127  00000FFC


PROC>register

-------------------------------------------------------------------------------
General Registers : sampled within : Section=001, Step=000
-------------------------------------------------------------------------------
r00 / 00000000 AAAAAAAA 000001B2 000001B2  00000004 00000005 00000006 00000007
r08 / 00000008 00000009 0000000A 0000000B  0000000C 0000000D 0000000E 0000000F
r16 / 00000010 00000011 00000012 00000013  00000014 00000015 00000016 00000017
r24 / 00000018 00000019 00000001 000EA000  00000000 0000001D 00000000 00054CCC
-------------------------------------------------------------------------------
```

```
PROC>reset;state

                          STATE VARIABLES
                          ---------------
                      (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print E   Isolation Print E
 Error Pause       S   Non-Error Pause E   Isolation Pause E
 Infinite Looping S    Hardcopy         S

 Section         0000000000    Step              0000000001
 Total loops     0000000001    Loops remaining   0000000001
 Step address      0005700C

 Selected Sections
 0-31    7FFFFFFF    32-63  FFFFFFFF
 64-95   FFFFFFFF   96-127  FFFFFFFC
```

Example 6. Bring the Diagnostic to a Known State

The example shows how to:

- Brings the diagnostic to a known default state: "reset;state".
- Sets infinite looping (loop)
- Confirms the status (state)

```
PROC>reset;state

                          STATE VARIABLES
                          ---------------
                      (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print E   Isolation Print E
 Error Pause       S   Non-Error Pause E   Isolation Pause E
 Infinite Looping S    Hardcopy        S

 Section         0000000000    Step              0000000001
 Total loops     0000000001    Loops remaining   0000000001
 Step address      00054C0C

 Selected Sections
 0-31    7FFFFFFF    32-63  FFFFFFFF
 64-95   FFFFFFFF   96-127  FFFFFFFC


PROC>loop

PROC>state

                          STATE VARIABLES
                          ---------------
                      (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print E   Isolation Print E
 Error Pause       S   Non-Error Pause E   Isolation Pause E
 Infinite Looping E    Hardcopy        S

 Section         0000000000    Step              0000000001
 Total loops     2147483647    Loops remaining   0000000001
 Step address      0005700C

 Selected Sections
 0-31    7FFFFFFF    32-63  FFFFFFFF
 64-95   FFFFFFFF   96-127  FFFFFFFC
```

Example 7. Using the "Loop" and "Resume" Commands

This example, shows the following:

- Re-initialize the diagnostic (reset command)
- Select sections 1 through 5 (section parameter)
- Examine the register contents prior to running the tests (register command)
- Execute the requested test sections once (resume command)
- Repeat the test five times (loop command).
- Display register values resulting from the test (register command)

Note that leaving out the "eeps" and "erronly" parameters forces display of each successive execution of all test sections and loop passes.

```
PROC> reset;state

                          STATE VARIABLES
                          ---------------
                    (E=ENABLED   S=SUPPRESSED )

 Error Print      E   Non-Error Print E   Isolation Print E
 Error Pause      S   Non-Error Pause E   Isolation Pause E
 Infinite Looping S   Hardcopy        S

 Section        0000000000    Step             0000000001
 Total loops    0000000001    Loops remaining  0000000001
 Step address     0005700C

 Selected Sections
 0-31    7FFFFFFF    32-63  FFFFFFFF
 64-95   FFFFFFFF   96-127  FFFFFFFC


PROC>section 1/5;state
```

```
                              STATE VARIABLES
                              ---------------
                          (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print E   Isolation Print E
 Error Pause       S   Non-Error Pause E   Isolation Pause E
 Infinite Looping S    Hardcopy         S

 Section         0000000000    Step              0000000001
 Total loops     0000000001    Loops remaining   0000000001
 Step address      0005700C

 Selected Sections
 0-31    7C000000    32-63  00000000
 64-95   00000000   96-127  00000000


PROC>register

-------------------------------------------------------------------------------
General Registers : sampled within : Section=001, Step=000
-------------------------------------------------------------------------------
r00 / 00000000 AAAAAAAA 000001B2 000001B2  00000004 00000005 00000006 00000007
r08 / 00000008 00000009 0000000A 0000000B  0000000C 0000000D 0000000E 0000000F
r16 / 00000010 00000011 00000012 00000013  00000014 00000015 00000016 00000017
r24 / 00000018 00000019 00000001 000EA000  00000000 0000001D 00000000 00054CCC
-------------------------------------------------------------------------------
PROC>resume

DIAGNOSTIC STARTED ON PROCESSOR# 0
   STARTING CPU DATA PATH TESTS - SECTIONS 1/6
001 002 003 004 005
PROC>loop 5;
PROC>resume

DIAGNOSTIC STARTED ON PROCESSOR# 0
   STARTING CPU DATA PATH TESTS - SECTIONS 1/6
001 002 003 004 005
DIAGNOSTIC COMPLETED ON PROCESSOR# 0
   END OF PASS 0000000001
DIAGNOSTIC STARTED ON PROCESSOR# 0
   STARTING CPU DATA PATH TESTS - SECTIONS 1/6
001 002 003 004 005
DIAGNOSTIC COMPLETED ON PROCESSOR# 0
   END OF PASS 0000000002
DIAGNOSTIC STARTED ON PROCESSOR# 0
   STARTING CPU DATA PATH TESTS - SECTIONS 1/6
001 002 003 004 005
DIAGNOSTIC COMPLETED ON PROCESSOR# 0
   END OF PASS 0000000003
DIAGNOSTIC STARTED ON PROCESSOR# 0
```

```
   STARTING CPU DATA PATH TESTS - SECTIONS 1/6
001 002 003 004 005
DIAGNOSTIC COMPLETED ON PROCESSOR# 0
   END OF PASS 0000000004
DIAGNOSTIC STARTED ON PROCESSOR# 0
   STARTING CPU DATA PATH TESTS - SECTIONS 1/6
001 002 003 004 005
   END OF PASS 0000000005
DIAGNOSTIC COMPLETED ON PROCESSOR# 0
   END OF PASS 0000000001
```

```
PROC>register
------------------------------------------------------------------------------
General Registers : sampled within : Section=005, Step=000
------------------------------------------------------------------------------
r00 / 00000000 000EC800 AAAAAAAA FFFFFFFF  D5555555 00000000 FFFFFFFF FFFFFFFF
r08 / 08000000 80000000 88888888 FFFFFFFF  7C000000 00000000 00000000 00000000
r16 / 00051200 00000011 00000012 00000001  000ECA80 00000000 58000004 00000001
r24 / FFF7E000 FFF7F034 00000005 000EA000  00000000 00100010 0001B830 00000100
------------------------------------------------------------------------------

PROC>state;register

                          STATE VARIABLES
                          ---------------
                     (E=ENABLED   S=SUPPRESSED )

 Error Print      E   Non-Error Print E   Isolation Print E
 Error Pause      S   Non-Error Pause E   Isolation Pause E
 Infinite Looping S   Hardcopy        S

 Section        0000000000    Step              0000000005
 Total loops    0000000005    Loops remaining   0000000000
 Step address     0005860C

 Selected Sections
 0-31    7C000000    32-63  00000000
 64-95   00000000   96-127  00000000


------------------------------------------------------------------------------
General Registers : sampled within : Section=005 Step=000
------------------------------------------------------------------------------
r00 / 00000000 000EC800 AAAAAAAA FFFFFFFF  D5555555 00000000 FFFFFFFF FFFFFFFF
r08 / 08000000 80000000 88888888 FFFFFFFF  7C000000 00000000 00000000 00000000
r16 / 00051200 00000011 00000012 00000001  000ECA80 00000000 58000004 00000001
r24 / FFF7E000 FFF7F034 00000005 000EA000  00000000 00100010 0001B830 00000100
------------------------------------------------------------------------------
```

Example 8. Diagnostic Failure / FRU Isolation

The example below shows the processor diagnostic failing with FRU/component isolation. Entering the "state" command displays the state variables of the failing test section:

```
PROC>reset;resume

DIAGNOSTIC STARTED ON PROCESSOR# 0
   STARTING CPU DATA PATH TESTS - SECTIONS 1/6
001 002 003 004 005 006
   STARTING SIU DATA PATH TESTS - SECTIONS 7/10
007
Error no. 010 Detected in Section 007

The fault is most likely in the SIU chip
```

```
PROC>state


                          STATE VARIABLES
                          ---------------
                     (E=ENABLED   S=SUPPRESSED )

 Error Print      E   Non-Error Print E   Isolation Print E
 Error Pause      S   Non-Error Pause E   Isolation Pause E
 Infinite Looping S   Hardcopy        S

 Section        0000000000    Step              0000000007
 Total loops    0000000001    Loops remaining   0000000001
 Step address     0005600C

 Selected Sections
 0-31    7FFFFFFF    32-63  FFFFFFFF
 64-95   FFFFFFFF   96-127  FFFFFFE0


PROC>register

--------------------------------------------------------------------------------
General Registers : sampled within : Section=007, Step=000
--------------------------------------------------------------------------------
r00 / 00000000 01200000 00000000 00056000  00000006 00054098 00AAAA0A FFFFFFFF
r08 / 55555555 80000000 88888888 FFFFFFFF  7FFFFFFF FFFFFFFF FFFFFFFF FFFFFE00
r16 / 00051200 00000001 00000000 00000001  000ECA80 00056018 58000004 00000001
r24 / FFF7E000 FFF7F034 00000007 000EA000  00000010 00100010 0001B830 00000100
--------------------------------------------------------------------------------
PROC>
```

Example Session: A1100AP SPU Processor Diagnostic

The following examples illustrate a diagnostic session using the A1100AP SPU Processor Diagnostic.

Example 1. Call up the Processor Diagnostic

```
ISL> a1100ap


Possible som_start found at 00022000.
System_id, a_magic = 020B0107.

Non-sharable executable SOM.
version_id = 05124000.
file_time = 0000000000000000.
entry_space, _subspace, _offset = 00000000, 00000000, 000506D0.
aux_header_location = 00000080.
aux_header_size = 00000030.
som_length = 000F2000.
aux_header_id = 10000004. aux_header_length = 00000028.
HP_UX exc_auxhdr.

DIAGNOSTIC SYSTEM FOR ISL STAND ALONE DIAGNOSTICS
Version 3.2


Type HELP for command information.


Type INFO for test information.


Series A1100AP System Processor Unit Diagnostic, Version 1.8
There are 126 Sections in the Diagnostic - 1 thru 126
```

Example 2. Using the State Command

This retrieves the initial state prior to running tests.

```
PROC>state

                          STATE VARIABLES
                          ---------------
                     (E=ENABLED   S=SUPPRESSED )

 Error Print      E   Non-Error Print E   Isolation Print E
 Error Pause      S   Non-Error Pause S   Isolation Pause E
 Infinite Looping S   Hardcopy        S

 Section        0000000000    Step              0000000000
 Total loops    0000000001    Loops remaining   0000000000
 Step address     00000000

 Selected Sections
 0-31    7FFFFFFF    32-63  FFFFFFFF
 64-95   FFFFFFFF   96-127  FFFFFFE0


PROC>seps


PROC>loop 1


PROC>sips
```

```
PROC>state


                          STATE VARIABLES
                          ---------------
                      (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print E   Isolation Print E
 Error Pause       S   Non-Error Pause S   Isolation Pause S
 Infinite Looping S    Hardcopy         S

 Section         0000000000    Step              0000000000
 Total loops     0000000001    Loops remaining   0000000001
 Step address      00000000

 Selected Sections
 0-31    7FFFFFFF    32-63  FFFFFFFF
 64-95   FFFFFFFF   96-127  FFFFFFE0
```

Example 3. Running the Program with Resume and Run

Resume restarts the diagnostic, with default test sections. State displays the system state.

```
PROC>resume


DIAGNOSTIC STARTED ON PROCESSOR# 0

   STARTING CPU DATA PATH TESTS - SECTIONS 1/6
001 002 003 004 005 006
   STARTING SIU DATA PATH TESTS - SECTIONS 7/10
007 008 009 010
   STARTING CCU0 DATA PATH TESTS - SECTIONS 11/18
011 012 013 014 015 016 017 018
   STARTING CCU1 DATA PATH TESTS - SECTIONS 19/26
019 020 021 022 023 024 025 026
   STARTING TCU DATA PATH TESTS - SECTIONS 27/40
027 028 029 030 031 032 033 034 035 036 037 038 039 040
   STARTING CPU INSTRUCTION TESTS - SECTIONS 41/93
041 042 043 044 045 046 047 048 049 050 051 052 053 054 055 056 057 058 059 060
061 062 063 064 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079 080
081 082 083 084 085 086 087 088 089 090 091 092 093
   STARTING CPU EXTENDED TESTS - SECTIONS 94/102
094 095 096 097 098 099 100 101 102
   STARTING FLOATING POINT TESTS - SECTIONS 103/126
103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
123 124 125 126
DIAGNOSTIC COMPLETED ON PROCESSOR# 0
   END OF PASS 0000000001
```

```
PROC>state

                          STATE VARIABLES
                          ---------------
                     (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print E   Isolation Print E
 Error Pause       S   Non-Error Pause S   Isolation Pause S
 Infinite Looping S    Hardcopy         S

 Section         0000000000    Step              0000000125
 Total loops     0000000001    Loops remaining   0000000000
 Step address      000F180C

 Selected Sections
 0-31    7FFFFFFF    32-63  FFFFFFFF
 64-95   FFFFFFFF   96-127  FFFFFFE0


PROC>exit

ISL>
```

Error Message Information

There are a number of error messages which may appear on the system console when the processor diagnostic is run. Each of these is described below.

`Error no. <errno> Detected in Section <secno>`

CAUSE This is the most common error message and usually means that the test (errno) in section (secno) failed.

ACTION

`Unexpected Trap# <trpno>, in Section <sectno>, Addr <addr>`

CAUSE This message occurs if a trap occurs which the test code is not trying to cause. When test code is trying to cause a specific trap, it loads the number of the trap into a register called trap_num (gr18). When the trap occurs, the trap handler checks the number of the trap that actually occurred with the number in trap_num. If they match then control is returned back to the test code where it occurred. Otherwise, control is transferred to code which prints out this message.

ACTION

`Bad Branch from Section# <secno>, into Section# <secno>`

CAUSE At the beginning of each test section, a procedure is called to print out the section number as an activity message. This also loads the section number into a local variable area at the label SECTNO. At the end of each test section, the section number is loaded into gr26 just before exiting back to seq_all (via the Break trap mechanism). After dumping the processor state, seq_all compares these two to see if control ended up in the same section it started in. If not then a wild branch is suspected and this message is printed.

ACTION

`Not enough unlocked space in CCU0 RAM, unable to test CCU1`

CAUSE Cache and CCU testing involves loading the test code into one cache bank, and then using it to test the other bank which is disabled so that code can not be executed from it. If the bank where the test code is to be loaded has locked lines where the test code would be loaded, then that code would hangup if any attempt is made to execute code from the address where the locked line is. Therefore the diagnostic must be aborted. The PSTAT command displays which lines are locked.

ACTION

CCU test code load into CCU0 RAM failed, unable to test CCU1

CAUSE After test code is loaded into one cache bank and before control is transferred to it, a check is made to see if the last address loaded is what it should be, and if no traps have occurred. If the loading process was not done properly, this message is printed.

ACTION

Cache RAM hard fault at <addr> in <field> bits: exp=<pat> act=<pat>

CAUSE This message appears if an error is found during the cache RAM test. The message indicates the location of the 1st address error, the field in which it was found (data, tag, or parity), and the expected and actual patterns. All cache lines with errors are locked out, and the PSTAT command can be used to display the locked lines.

ACTION

There are <number> locked-out entries, starting at <addr>

CAUSE This message appears after the RAM hard fault message above and indicates the number of lines with hard errors.

ACTION

DIAGNOSTIC ABORTED because of locked lines in other CCU

CAUSE After all the cache tests are run, a check is made to see if either of the CCU's had failed any of the tests, other than the RAM tests. If so, that CCU is disabled and the other cache bank RAM is tested to see if it has any locked entries. If it does, the diagnostic must be aborted because only one bank is enabled and the machine hangs up if an attempt is made to access the locked line.

ACTION

Both Cache Entries at <addr> are locked out, Diagnostic Aborted

CAUSE After both cache banks have been tested, and both CCU's have passed, the lock status of both banks is checked. If any cache lines at the same entry addresses in both banks are locked, then the machine can hangup if that address is accessed, so this message is printed and the diagnostic is aborted.

ACTION

TLB RAM hard fault at <addr> in <field> field, exp= <pat>, act=<pat>

CAUSE The TCU test sections test all the fields in the TLB tag RAMs. These include RPN, SID, VPN, PID, Access Rights, and the Valid Bit and Flag fields. If an error is found by these tests, then this message is printed to indicate the address where the error was detected, the name of the field, and the expected and actual patterns. For fields shorter than the maximum, the relevant bits are left justified. Parts of the TLB are also locked out; these can be determined by observing the Lock Value bits (16..18) and Lock Enable bits (19..21) in TCU Diagnose register 0, using PSTAT to display its contents.

ACTION

All TLB RAM Sectors faulty in <field>, test aborted, exp=<pat> act=<pat>

CAUSE If the TLB RAM tests find errors in all 8 sectors of the TLB, causing the entire TLB to be locked out, then this message is printed, instead of the previous one above, and the remainder of the TCU tests are bypassed, since no further tests are possible.

ACTION

All test patterns fail in <field>, starting at <addr>

CAUSE If all test patterns fail in the TLB RAM tests, this message is printed to indicate the widespread problem.

ACTION

Contents

Tables

4

SPU Memory Diagnostic (A1002AM, A1100AM)

Introduction

The HP A1002A/A1100A SPU Memory Diagnostic is intended to reduce the isolation time for bad Memory Controllers and/or bad Memory Arrays.

Systems Tested by the A1002AM SPU Memory Diagnostic

The following table lists all current system families tested by the A1002AM SPU Memory Diagnostic.

Table 4-1. Systems Tested by A1002AM SPU Memory Diagnostic

Family	Label	Code Name	Models
825/925	x25	Firefox	825, 925
	x35	Topgun	835, 935, etc.
	x45	Shogun	845, 949, etc.

Systems Tested by the A1100AM SPU Memory Diagnostic

The following table lists all current system families tested by the A1100AM SPU Memory Diagnostic.

Table 4-2. Systems Tested by A1100AM SPU Memory Diagnostic

Family	Label	Code Name	Models
850/950	x50	Cheetah	850S, 950, etc.
	x55	Jaguar	855S, 955, etc.
	x60	Cougar	960
	x80, x70	Panther	870, 980

Note

The code names listed in the two preceding tables are not official; however you may see them in internal documents or hear them used by factory personnel.

Chapter Organization

This chapter provides the following information about the SPU Memory diagnostic:

- Functional Description
- Test Section Descriptions
- Test Sequence
- Example Session: A1002AM SPU Memory Diagnostic
- Example Session: A1100AM SPU Memory Diagnostic
- Error Message Explanations

Defects and Enhancements

Submit defect reports and enhancement requests for this diagnostic through the STARS database, referencing product number 30342-10002 (A1002AM) and 30343-10002 (A1100AM).

Minimum Configuration

The minimum hardware configuration required to load and run the offline diagnostics system consists of the following functional hardware, firmware, and software items:

- HP Precision Architecture RISC SPU with Main Memory and Processor Dependent Code (PDC)
- Boot device and boot path hardware
- System console
- Initial System Load Code (ISL)
- Offline Diagnostics Software

Functional Description

The user is strongly urged to carefully read and understand this information before attempting to use this diagnostic for the first time. Online help is available to the user through use of the ISL based Offline Diagnostics Environment "Help" facility.

The SPU Memory Diagnostic performs two kinds of tests: functional and parametric.

- Functional tests establish the presence or absence of a memory function.
- Parametric tests assess a memory function in relation to some performance or environmental factor.

An example of functional testing is checking for the presence of a short or open circuit. Parametric testing involves such things as reading or writing data over a variety of speed, temperature, and voltage ranges and then checking the performance of these functions. Another example of parametric testing is checking for the presence or absence of parasitic resistances or capacitances that are out of specification.

Three main hardware areas are checked by the diagnostic: MID_BUS lines, Memory Controller, and Memory Array. The diagnostic test sections which correspond to these hardware areas are defined below.

The memory diagnostic works through two primary communication pathways within the overall hardware environment. The stimulus-response path which generates test vectors moves on the path: processor >> cache >> MID_BUS >> memory controller >> memory array bus >> memory array. The command message path for the user interface moves on the path: processor >> MID_BUS >> CIO adapter >> I/O system. Memory arrays may be 2 Mbytes, 8 Mbytes, 16 Mbytes, or 64 Mbytes.

The test requires approximately 1 minute per 16Mbyte memory array with all test sections enabled.

Unique Commands

The memory diagnostic supports two commands not found in the UI: CARD and DECODE.

CARD	*{space / integer:0 ... 31 / integer integer /integer/integer / [integer] [integer/integer]}*. Permits selecting the memory array to be tested by setting a 32-bit bit mask. It also performs range checking on the integer(s) supplied. If an out of range integer is supplied, then the comand is rejected and the message **`Parameter out of range`** is displayed. Memory arrays 0-7 correspond to controller 0, 8-15 correspond to controller 1. If the first parameter exceeds the second, the message **`Illegal order`** is displayed. When no parameter is provided, the card select bit mask value is displayed in hexadecimal.
DECODE	*{integer: 0 ... FF}*. Permits converting an error syndrome code into a single bit error, double bit error or no error. It produces one of three messages: **`No Error`**, **`Single bit error in bit position NN`**(NN may range from 00 to 63), **`Double bit error`**. An error syndrome code is the checksum difference between the bits stored and generated during the read.

Test Section Descriptions

The organization and coverage of the SPU Memory Diagnostic is arranged by test section number in ascending order. There are a total of 18 sections. Table 4-1 below lists these tests sections according to number, name, whether or not the test is functional or parametric, and the major hardware area tested. Following the table each test is explained in detail.

Table 4-1. A1002AM Memory Diagnostic Organization and Coverage

Section #	Test Name	Test Class	Hardware Area
1	MID_BUS lines	Functional	MID_BUS
2	Read, Hammer, Write	Parametric	MID_BUS
3	Write, Hammer, Read	Parametric	MID_BUS
4	Data Square Wave	Parametric	MID_BUS
5	Semaphore Wave	Functional	Mem. Cntrlr.
6	Single Bit Error	Functional	Mem. Cntrlr.
7	Force Syndrome	Functional	Mem. Cntrlr.
8	Double Bit Error	Functional	Mem. Cntrlr.
9	Double Bit Parametric	Parametric	Mem. Cntrlr.
10	Directed Reset	Functional	Mem. Cntrlr.
11	Cas Only	Functional	Mem. Array
12	RAM Address Lines	Functional	Mem. Array
13	RAM Chips	Functional	Mem. Array
14	Read, Hammer, Write	Parametric	Mem. Array
15	Write, Hammer, Read	Parametric	Mem. Array
16	Column Strife Test	Parametric	Mem. Array
17	Static Soccer	Parametric	Mem. Array
18	Moving Inversions	Both	MC. and MA.

The test sections are described below:

Section	Description
1	**MID_BUS Lines** Tests for shorts and opens by writing and reading walking ones and zeros to COMMAND register. (functional)
2	**Read,Hammer,Write** Initialize the SPA and COMMAND registers to 0. Then the SPA register is read at maximum speed for 10ms. Then a single write is made to the COMMAND register. Then the SPA and COMMAND registers are read to assure that the write worked correctly. (parametric)
3	**Write,Hammer,Read** Initialize the SPA and COMMAND registers to 0. Then the COMMAND register is written at maximum speed for 10ms. Then a single read is made from the SPA register, and the SPA and COMMAND registers are read to assure that the writes worked correctly. (parametric)
4	**Data Square Wave** Tests the transient response of the Memory Controller Data Drivers and Receivers. The following data patterns are written and read to the COMMAND register for 10ms. (parametric) 00000000. FFFFFFFF. 00000000. FFFFFFFF.
5	**Semaphore Wave** Tests that a cache line modification can be flushed back to memory. The algorithm is write, semaphore, read. (functional)
6	**Single Bit Error** Generates and tests all single bit data failures using the FORCE_ERROR registers of the MC. The four double words of the error registers are initially set to X'00000000 00000001. The cache line is written and read from memory, the cache line is checked for the proper four double patterns and status is checked for expected results. The four double words of the error registers are shifted by one position for each iteration of the code until the test double words become X'80000000 00000000. (functional)

Section	Description
7	**Force Syndrome** Verifies the operation of the Hamming circuits and the FORCE_SYNDROME registers by flipping a bit in the returned syndrome. It tests single bit errors in the parity portion of the memory word(bits 64-71). The data is be read and checked. Error logging is also checked. (functional)
8	**Double Bit Error** Generates a single bit error in memory. The error is read and the information returned is used to set up the IO_MAP_ADDRESS and IO_MAP_SYNDROME registers. Then FORCE_ERROR registers are set up to force a double bit error, one bit is the same as above. MC should correct the bit that has been mapped out and report a single bit error for the remaining value. Finally, a double bit error is sent without mapping it out. This check is for an HPMC to occur. (functional)
9	**Double Bit Parametric** Generates a single bit error in memory. The error is read and the information returned is used to set up the IO_MAP_ADDRESS and IO_MAP_SYNDROME registers. Then FORCE_ERROR registers are set up to force a double bit error, one bit is the same as above. This process is repeated at maximum speed for 10ms, and the status is checked to verify that a single bit error was detected and corrected. (parametric)
10	**Directed Reset** Issues a directed reset against the memory controller. It then reads all of the memory controller status and restores the memory controller status as much as possible. Finally, this section compares the reset status against the expected status. (functional)
11	**CAS only Marching Ones and Zeros** Uniquely sets and clears each bit in the CAS data latch, by setting up the following patterns in memory. It is assumed that the CAS data latch is 2048 bits wide for 1Mbit RAMs and 1024 bits wide for 256K RAM's. (functional)

```
   10000000...                  01111111...
R  01000000...               R  10111111...
A  00100000...               A  11011111...
S  00010000...               S  11101111...
   ...                          ...
   CAS                          CAS
```

12	**Array Address Lines** Simplified marching ones and zeros. Tests the uniqueness of addresses. The algorithm is: Initialize the RAM with a known value. Read the first address, Compare, Complement the data in the first address; read it back; and compare. Read the second address, and so on. After the last address has been tested, the entire memory has been complemented. Now begin the algorithm all over except decrement the address. (functional)

Section	Description
13	**RAM chips** Every location should be written as a one and as a zero including the syndrome bits. This section employs two tests - referred to as the sliding-ONE test and the sliding-ZERO test - that contain four parts each. All of the parts work similarly, but test for different types of faults. The algorithm is: Initialize memory to low values. Read the first address. Compare. Write a binary pattern in which all bits except the least significant bit are set to ZERO. Read the second address. Compare. Write a word wherein all bits except the second-from-right bit are cleared. Read all locations when done to verify. Part 2 of this test fills memory in reverse order. Parts 3 and 4 are the same as 1 and 2 except memory is initialized high. The sliding-ZERO test does the same four parts but slides a zero through all ones in the pattern. (functional)
14	**Read, Hammer, Write** Tests the transient response of the chip address logic. The RAM is initialized to known random values. The lowest available address is repetitively read at maximum speed for 10ms, and a single write is made to the maximum address. The whole RAM is then read to assure that the write worked correctly. (parametric)
15	**Write, Hammer, Read** Similar to Read Hammer Write, except that this test uses writes as the repetitive function. (parametric)
16 & 17	**Tests for data persistence (Column Strife & Static Soccer Test)** Parametric tests that check for out-of-bounds parasitic resistances. If the parasitic resistance between memory cells is too low, then the value stored in one cell influences the value read from a nearby cell. The general algorithm is to initialize a RAM to a known value which accentuates data persistence problems. The diagnostic pauses while the cells influence one another; then one quickly checks the RAM for data integrity. The key factors are the data pattern and the wait time in relationship to the refresh interval. The first pattern shown below is for the Column Strife Test, the second pattern is for the Static Soccer Test. (parametric)

```
Column Strife Test:

    The following pattern is set up in memory and held for 10ms.:
          010101010101010101 ....
    R     010101010101010101 ....
    A     010101010101010101 ....
    S     010101010101010101 ....
          .....
          CAS

Static Soccer Ball:

    The following pattern is set up in memory and held for 10ms.:
          000000000000000000 ....
    R     010101010101010101 ....
    A     000000000000000000 ....
    S     010101010101010101 ....
          .....
          CAS
```

Section	Description
18	**Moving Inversions** Enhance the marching ones and zeros test by introducing the concept of a carry coupled adder for the address function; the carry out from the highest order bit becomes the carry in of the lowest order bit. Initially, the inversion test is the same as the marching ones test: Initialize the RAM to zero. Read the first address, address zero. Compare. Complement the data in the first address. Compare. Read the second address, address one. After the last address has been tested the entire memory has been complemented. The second pass of the test also starts at address zero but the address is now incremented by two until carry roll over occurs. So one first tests the even address and then tests the odd addresses. The third pass one increments the address by four. The first three address patterns are: 1. 0 1 2 3 4 5 6 7 8 ... 0 2. 0 2 4 6...1 3 5 7 ... 0 3. 0 4 8 C...1 5...2 6...3 7...0 Each new address modifier is twice the size of the previous address modifier; 1, 2, 4, 8, etc. The test is stopped when the first address pattern would repeat. This is a lengthy test that runs approximately 3.5 hours for a 2Mbyte board. Due to its length, it displays step numbers 1-129 in about 10 minutes, then displays the section number again. (parametric and functional)

Table 4-2. A1100AM Memory Diagnostic Organization and Coverage

Section #	Test Name/Function	Hardware Area
0	Performs CRC check of diagnostic code.	
1	Walking Ones and Zeroes (dp).	First Mem. Cntrlr.
2	Simple address test of RAS/CAS lines.	First Mem. Cntrlr.
3	Full available address range test off first MC.	First Mem. Cntrlr.
4	All single bit errors.	First Mem. Cntrlr.
5	Single bit error logging to syndrome register.	First Mem. Cntrlr.
6	Double bit error test.	First Mem. Cntrlr.
7	Directed reset to the MC.	First Mem. Cntrlr.
8	Force MAB parity Error on read.	First Mem. Cntrlr.
9	Force MAB parity Error on write.	First Mem. Cntrlr.
10	Force SMB write data parity error.	First Mem. Cntrlr.
11	Powerfail and Soft Error Stimulus.	First Mem. Cntrlr.
12	Powerfail Check.	First Mem. Cntrlr.
13	Soft Error Check.	First Mem. Cntrlr.
14	Hole Filling Test.	First Mem. Cntrlr.
15	Walking Ones and Zeroes.	Second Mem. Cntrlr.
16	Simple address test of RAS/CAS lines.	Second Mem. Cntrlr.
17	Full address test of available address range off second MC.	Second Mem. Cntrlr.
18	All single bit errors.	Second Mem. Cntrlr.
19	Single bit error logging to syndrome register.	Second Mem. Cntrlr.
20	Double bit error test.	Second Mem. Cntrlr.
21	Directed reset to the MC.	Second Mem. Cntrlr.
22	Force MAB parity Error on read.	Second Mem. Cntrlr.
23	Force MAB parity Error on write.	Second Mem. Cntrlr.
24	Force SMB write data parity error.	Second Mem. Cntrlr.
25	Hole Filling Test.	Second Mem. Cntrlr.
26-29	Expansion Space.	Second Mem. Cntrlr.

Table 4-2. A1100AM Memory Diagnostic Organization and Coverage (continued)

Section #	Test Name/Function	Hardware Area
30	Data Square Wave Test.	Mem. Array
31	Semaphore Square Wave Test.	Mem. Array
32	CAS only Marching One's test.	Mem. Array
33	Marching one's test.	Mem. Array
34	Read Hammer Write test.	Mem. Array
35	Write Hammer Read.	Mem. Array
36	Address Square Wave Test.	Mem. Array
37	Column Strife test of data persistence.	Mem. Array
38	Static Soccer Ball Test of data persistence.	Mem. Array
39	Moving Inversion Test.	Mem. Array
50	Menu Driven Section.	Mem. Array

Test Sequence

To select the Memory Diagnostic, type "A1002AM" or "A1100AM". After the ISL banner appears, the diagnostic identification banner appears, as shown below:

```
HP A1002A/A1035A Memory Diagnostic, Version 1.3

The HPA of the processor is FFF80000
The processor model is A.0.4.01.
```

This is followed by the diagnostic prompt "MEM>" and the system then waits for user commands. At this point select which sections and other parameters. Hardcopy printout can be selected at this point (Not supported on the A1002AM SPU Memory Diagnostic). Default parameters are given below.

Parameter	State
default sections	1/17
activity indicators	enabled
error and isolation messages	enabled
pause after error and isolation message	enabled
looping or hardcopy	no

To begin execution, type "RESUME". See chapter 3 for command options and syntax. The first message to appear is:

```
SECTION 001
000 001
        (ETC.)
          :
          :
```

The following message appears after the last test has been executed:

```
DIAGNOSTIC COMPLETED
MEM>
```

The test requires 1 to 2 minutes to complete. If looping has been enabled, at the end of each pass through the diagnostic, a message of the following form is displayed:

```
END OF PASS 0000000001
```

If error messages have been enabled and a fault is detected, a message in the following form is displayed:

```
ERROR IN SECTION 012, Step 001.
ERROR at aaaaaaaa. WAS bbbbbbbb. SHOULD BE cccccccc.
```

"aaaaaaaa" is the address at which the error was detected. Values "bbbbbbbb" and "cccccccc" are the actual and expected data respectively.

Exceptions to this arrangement are described in the error message information section located at the end of this chapter.

Run and Resume

It **Resume** has not been entered already, the **Run** command displays the banner and then halts at the MEM> prompt. If **Resume** has been entered, **Run** executes the diagnostic.

The **Resume** command executes the diagnostic with the selected parameters.

Section 50 - Interactive Menu Driven Section

This section allows you run a variety of different memory tests via a menu-driven interface:

```
                    A1100AM Menu

    1. Print Memory Card Configuration.
    2. Single array test.
    3. Comprehensive test of Memory Controllers.
    4. Short test of all memory.
    5. Medium test of all memory.
    6. Comprehensive test of all memory.
    7. Exit this menu driven section to diagnostic prompt.
    8. Exit this diagnostic and return to ISL.

    Enter one of the above numbers (1 - 8) >
```

Example Session: A1002AM SPU Memory Diagnostic

This set of examples illustrates a diagnostic session using the HP A1002A SPU Memory Diagnostic. The dialog below shows the user first calling up the diagnostic which responds with the appropriate header information and a table of configured memory controllers. The "help" command is entered which displays a summary of all available commands. Next, the information command is used to display the test sections. The state command is issued to assess the diagnostic system state variables prior to actually running the diagnostic. The reset and resume commands then actually run the diagnostic.

Example 1. Call Up Memory Diagnostic

This portion of the example illustrates how a user calls up the memory diagnostic. Some output is omitted at the end of the display.

```
ISL> A1002AM

Beginning of SOM address = 0x00039000
Beginning of code space address = 0x00040000
Last code space address = 0x000517FC
Last SOM address = 0x00058E20
DIAGNOSTIC SYSTEM FOR ISL STAND ALONE DIAGNOSTICS
Version 1.0

Type HELP for command information.

Type INFO for test information.

HP A1002A/A1035A Memory Diagnostic, Version 1.3
The HPA of the processor is FFF80000.
The processor model is A.0.4.01.

 Table of Memory Controllers in system
module:32; HPA: FFFA0000; SPA: size = 2^23 = 8388608; Model: 2.F.8.00.
. . .

MEM>
```

Example 2. Obtaining Help and Information

Next the "help" and "information" commands are entered to display the command keywords and what tests are available.

```
MEM>help

         Command Keyword Repertoire

 SECTION, LOOP, RESUME, RESET, EEPS, SEPS, EIPS,
 SIPS, ENPS, SNPS, ERRPAUSE, EEPR, SEPR, EIPR,
 SIPR, ENPR, SNPR, ERRONLY, HARDCOPY, STATE,
 STOP, LISTIO, CHANGEIO, HELP, INFORMATION,
 DEBUG, DUMP, REGISTER, RUN, EXIT

 enter Help <keyword> for discrete descriptions.

MEM>information

    Sections Available In This Diagnostic

 1=MidBus Lines          2=Read,Hammer,Write
 3=Write,Hammer,Read     4=Data Square Wave
 5=Semaphore Wave        6=Single Bit Error
 7=Force Syndrome        8=Double Bit Error
 9=Double Bit Para.     10=Directed Reset
 11=Cas Only            12=RAM Address lines
 13=RAM chips           14=Read,Hammer,Write
 15=Write,Hammer,Read   16=Column Strife Test
 17=Static Soccer       18=Moving Inversions

 enter INFO <number> for discrete descriptions.
```

Example 3. Checking System State Variables

Before running the entire diagnostic, make a check of the Offline Diagnostic system state variables by entering the state command:

```
MEM>state

                          STATE VARIABLES
                          ---------------
                    (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print E   Isolation Print E
 Error Pause       E   Non-Error Pause S   Isolation Pause E
 Infinite Looping S    Hardcopy         S

 Section         0000000000    Step              0000000000
 Total loops     0000000001    Loops remaining   0000000000
 Step address      00000000

 Selected Sections
 0-31    7FFFC000    32-63  00000000
 64-95   00000000   96-127  00000000
```

Example 4. Using Reset and Resume

The diagnostic is initialized by the "reset" command and runs when the user enters "resume":

```
MEM>reset

MEM>resume

SECTION 001
000
 Limited to sections 5,aa-17 on this MC
 Changing section map


SECTION 005
000 001

SECTION 011
000 001 002

SECTION 012
000 001 002 003 004

SECTION 013
000 001 002 003 004

SECTION 014
000 001 002

SECTION 015
000 001 002

SECTION 016
000 001 002

SECTION 017
000 001 002

DIAGNOSTIC COMPLETED
MEM>
```

Example Session: A1100AM SPU Memory Diagnostic

Example 1. Calling Up the Diagnostic

```
ISL> a1100am

Possible som_start found at 0003A800.
System_id, a_magic = 020B0107.

Non-sharable executable SOM.
version_id = 05124000.
file_time = 0000000000000000.
entry_space, _subspace, _offset = 00000000, 00000000, 000406B8.
aux_header_location = 00000080.
aux_header_size = 00000030.
som_length = 00023000.
aux_header_id = 10000004. aux_header_length = 00000028.
HP_UX exc_auxhdr.

DIAGNOSTIC SYSTEM FOR ISL STAND ALONE DIAGNOSTICS
Version A.xx.xx                    This is not the diagnostic version number
Type HELP for command information.
Type INFO for test information.
Undecoded (raw) Console Path is 255 255 255 255 255 2 4

MEM>
```

Example 2. Obtaining Information

```
MEM>info

The test functionality of the individual test sections is described below:

Sect.   Description
------------------------------------------------------------
 0 Performs CRC check of diagnostic code.

 Test sections 1-14 test the first Memory Controller.

 1 Walking Ones and Zeroes (dp).
 2 Simple address test of RAS/CAS lines.
 3 Full address test of available address range off first MC.
 4 All single bit errors.
 5 Single bit error logging to syndrome register.
 6 Double bit error test.
 7 Directed reset to the MC.
 8 Force MAB parity Error on read.
 9 Force MAB parity Error on write.
10 Force SMB write data parity error.
11 Powerfail and Soft Error Stimulus. (see note below)
12 Powerfail Check.                   (see note below)
13 Soft Error Check.
14 Hole Filling Test.

 Test sections 15-29 test the second Memory Controller.

15 Walking Ones and Zeroes.
16 Simple address test of RAS/CAS lines.
17 Full address test of available address range off second MC.
18 All single bit errors.
19 Single bit error logging to syndrome register.
20 Double bit error test.
21 Directed reset to the MC.
22 Force MAB parity Error on read.
23 Force MAB parity Error on write.
24 Force SMB write data parity error.
25 Hole Filling Test.
26-29 Expansion Space.
```

```
Test sections 30-39 test the Memory Arrays.

30 Data Square Wave Test.
31 Semaphore Square Wave Test.
32 CAS only Marching One's test.
33 Marching one's test.
34 Read Hammer Write test.
35 Write Hammer Read.
36 Address Square Wave Test.
37 Column Strife test of data persistence.
38 Static Soccer Ball Test of data persistence.
39 Moving Inversion Test.
50 Menu Driven Section.

The diagnostic also has one diagnostic specific command:

CARD {integer:0...15| [integer] [integer/integer]};

MEM>
```

Note

The DECODE command is now obsolete, because the diagnostic can now diagnose down to the U number of the bad chip.

Note

The system console path must be changed to 16.0.0 prior to running sections 11 and 12. See the description in the section: "Power Fail Recovery Test".

Example 3. Running Selected Sections

```
MEM>state

                          STATE VARIABLES
                          ---------------
                     (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print E   Isolation Print E
 Error Pause       S   Non-Error Pause S   Isolation Pause E
 Infinite Looping S    Hardcopy         S

 Section         0000000000    Step              0000000000
 Total loops     0000000001    Loops remaining   0000000000
 Step address      00000000

 Selected Sections
 0-31     80000000    32-63  00000000
 64-95    00000000   96-127  00000000

MEM>section 1/100

MEM>seps

MEM>state


                          STATE VARIABLES
                          ---------------
                     (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print E   Isolation Print E
 Error Pause       S   Non-Error Pause S   Isolation Pause E
 Infinite Looping S    Hardcopy         S

 Section         0000000000    Step              0000000000
 Total loops     0000000001    Loops remaining   0000000000
 Step address      00000000

 Selected Sections
 0-31     7FE00000    32-63  00000000
 64-95    00000000   96-127  00000000
```

Example 4. Using the Resume Command

The `resume` command, when used without the `section` command, will run section 0 (the default) only. To specify other test sections instead of, or in addition to, section 0, use the `section` command (see `info` for details). The following is a list of available test sections and their descriptions:

Section 0	Software check
Sections 1-10	First memory controller
Sections 11-12	Power fail test
Section 13	Soft error rate check
Sections 15-25	Second memory controller
Sections 30-39	Memory arrays for both memory controllers
Section 50	Interactive menu section

Section 0 is a diagnostic software integrity check. If section 0 fails, possible causes are bad software (i.e., a bad support tape) or a fault elsewhere in the system:

- CPU
- I/O
- Memory controller 0
- Memory array 0

If section 0 fails, replace the suspect tape with a known good tape, and run the test again.

Memory array configuration information is presented in the form `MA card #`*n* `not present` or `MA card #`*n* `is a` *n*`Mb card`; when troubleshooting, it is a good idea to check this information to see if it agrees with the physical configuration and to see if this is a legal configuration.

Caution 

If a message in the general form `Second MC not found. PDC status = 1.` is displayed, do not run any second controller tests.

If a second controller is present, the memory configuration information will be presented in the same format as for the first memory controller.

The following example demonstrates a typical use of the **resume** command, with some sample values.

```
MEM>card 0 8  select memory array cards to be tested
MEM>section 0/39  select test sections 0-39, inclusive, to be run
MEM>resu          run the diagnostic test sections selected or the default if none selected

HP A1100A SPU Memory Diagnostic
Part Number 31090-90020. Version A.09.02

The HPA of the processor is FFF8A000.

The Hversion = 00000830. The Sversion = 00000413.
PDC_CACHE values are:
I_size = 00080000. I_conf = 01412000. I_base = 00000000.
I_stride = 00000020. I_count = 00002000. I_assoc = 00000002.

SECTION 000

 * Computing CRC on diagnostic SOM.

The time taken for section 0 is less than 1 second.


The IODC of the first MC is 00441B41, 00000900.

First Memory Controller Status:

Reg  9: 00000020 :SPA
Reg 13: 00026440 :Status; DE, ME = 00000000
Reg 14: 00000100 :Control; Enable Error Corr. = 00000100
Reg 16: 00000620 :Error Address
Reg 17: FFFF0097 :Error Syndrome
Reg 32: 07C00000 :IO_MBMASK0
Reg 33: 07C00000 :IO_MBMASK1
Reg 34: 00001000 :IO_MBCONF0
Reg 35: 00000000 :IO_MBCONF1
Reg 36: 80180020 :IO_SPAMASK
Reg 48: 00008000 :Memory Configuration
MA slot #0: 16 Mb board.  configuration information
MA slot #1: empty.
MA slot #2: empty.
MA slot #3: empty.
MA slot #4: empty.
MA slot #5: empty.
MA slot #6: empty.
MA slot #7: empty.
The key values are:
Maximum memory = 33554432. (H) = 02000000.
Maximum SPA = 33554432. (H) = 02000000.
IO_MBMASK0, IOMBMASK1 = 07C00000 : 07C00000
```

```
IO_MBCONF0, IO_MBCONF1 = 00001000 : 00000000
IO_SPAMASK = 80180020.


The IODC of the second MC is 00441B41, 00000900.

Second Memory Controller Status:

Reg  9: 00000020 :SPA
Reg 13: 00026440 :Status; DE, ME = 00000000
Reg 14: 80000100 :Control; Enable Error Corr. = 00000100
Reg 16: 20000620 :Error Address
Reg 17: FFFF0097 :Error Syndrome
Reg 32: 07C00000 :IO_MBMASK0
Reg 33: 07C00000 :IO_MBMASK1
Reg 34: 00001000 :IO_MBCONF0
Reg 35: 00000000 :IO_MBCONF1
Reg 36: 80180020 :IO_SPAMASK
Reg 48: 00008000 :Memory Configuration
MA slot #8: 16 Mb board. configuration information
MA slot #9: empty.
MA slot #10: empty.
MA slot #11: empty.
MA slot #12: empty.
MA slot #13: empty.
MA slot #14: empty.
MA slot #15: empty.
The key values for the second MC are:
Maximum memory = 33554432. (H) = 02000000.
Maximum SPA = 33554432. (H) = 02000000.
```

```
SECTION 000

 * Computing CRC on diagnostic SOM.

The time taken for section 0 is less than 1 second.


SECTION 001
Walking 1's and 0's Across MAB. Data path Logic test.

 *  *  *
Disabling ECC.
 *  *
Enabling ECC.

The time taken for section 1 is 0 minutes and 1 seconds.


SECTION 002
Simple Address Test of First MC.

 *
The time taken for section 2 is 0 minutes and 11 seconds.


SECTION 003
Full address test of first MC.

 *
Address = 00200000  Address = 00400000  Address = 00600000  Address = 00800000
Address = 00A00000  Address = 00C00000  Address = 00E00000  Address = 01000000
Address = 01200000  Address = 01400000  Address = 01600000  Address = 01800000
Address = 01A00000  Address = 01C00000  Address = 01E00000
Flushing Processor cache. Compare follows.

The time taken for section 3 is 0 minutes and 11 seconds.


SECTION 004
Single Bit Error Test.

 *  *  *
The time taken for section 4 is less than 1 second.


SECTION 005
Single Bit Error Logging Logic Test.

 *
The time taken for section 5 is less than 1 second.
```

```
SECTION 006
Double Bit Error Test: i_vector_table = 00043800. mc_hpmc = 0004A414.
 *
The time taken for section 6 is less than 1 second.


SECTION 007
Direct Reset of Single Bit Error Test.

 *
The time taken for section 7 is less than 1 second.


SECTION 008
MAB Read Parity Error Test. pim_buf = 000170C0. r_addr = 000172C0.

 *
The time taken for section 8 is 0 minutes and 1 seconds.


SECTION 009
Directed Reset of MAB Write Parity Error Test.
pim_buf = 000170C4. r_addr = 000172C4.

 *
The time taken for section 9 is less than 1 second.


SECTION 010
SMB EDC Test.

 *
The time taken for section 10 is less than  1 second.


SECTION 011

First part of powerfail test.

Direct port is not configured in both stable storage and page 0.

Switching console path to PDH DIRECT PORT.  Takes about one minute...
 Loading ENTRY_INIT.
        Executing ENTRY_INIT.
 Loading ENTRY_IO.
Console successfully switched to PDH DIRECT PORT.

 *
The time taken for section 11 is 1 minutes and 20 seconds.
```

```
SECTION 012
Second part of powerfail test.  Shut of AC power now!!!!!!!!!!!

 *
The time taken for section 12 is 0 minutes and 5 seconds.


SECTION 013
Soft Error Check.

 *
The time taken for section 13 is 0 minutes and 5 seconds.


SECTION 014
Hole Filling Test of first Memory Controller.
Hole Filling Test runs only if interleaving is off.

The time taken for section 14 is 0 minutes and 1 seconds.


SECTION 015
Walking 1's and 0's Across MAB. Data path Logic test.

 *  *  *
Disabling ECC.
 *  *
Enabling ECC.

The time taken for section 15 is less than 1 second.


SECTION 016
Simple Address Test of Second MC.  Base = FA1AD320.

 *
The time taken for section 16 is less than 1 second.
SECTION 017
Full address test of second MC.

 *
Address = 00800000  Address = 01000000  Address = 01800000
Flushing Processor cache. Compare follows.

The time taken for section 17 is 0 minutes and 9 seconds.
```

```
SECTION 018
Single Bit Error Test.

 *  *  *
The time taken for section 18 is less than 1 second.


SECTION 019
Single Bit Error Logging Logic Test.

 *
The time taken for section 19 is less than 1 second.


SECTION 020
Double Bit Error Test: i_vector_table = 00043800. mc_hpmc = 0004A414.
 *
The time taken for section 20 is less than 1 second.


SECTION 021
Direct Reset of Single Bit Error Test.

 *
The time taken for section 21 is 0 minutes and 1 seconds.


SECTION 022
MAB Read Parity Error Test. pim_buf = 000170C0. r_addr = 000172C0.

 *
The time taken for section 22 is less than 1 second.


SECTION 023
Directed Reset of MAB Write Parity Error Test.
pim_buf = 000170C4. r_addr = 000172C4.

 *
The time taken for section 23 is 0 minutes and 1 seconds.


SECTION 024
SMB EDC Test.

 *
The time taken for section 24 is less than  1 second.
```

```
SECTION 025
Hole Filling Test of second Memory Controller.
Hole Filling Test runs only if interleaving is off.

The time taken for section 25 is 0 minutes and 1 seconds.


dummy section
dummy section
dummy section
dummy section


MA_test card no. = 0.

SECTION 030
Data Square Wave test. Base address = 0006B4C0.

 *
The time taken for section 30 is 0 minutes and 1 seconds.

--------------------------------------------------------------------------------

MA_test card no. = 8.

SECTION 030
Data Square Wave test. Base address = 0006B4E0.

 *
The time taken for section 30 is 0 minutes and 1 seconds.

--------------------------------------------------------------------------------

MA_test card no. = 0.

SECTION 031
Semaphore Wave test.  Base address = 0006B4C0.

 *
The time taken for section 31 is 0 minutes and 1 seconds.

--------------------------------------------------------------------------------
```

```
MA_test card no. = 8.

SECTION 031
Semaphore Wave test.  Base address = 0006B4E0.

 *
The time taken for section 31 is 0 minutes and 1 seconds.
-------------------------------------------------------------------------------
MA_test card no. = 0.

SECTION 032
CAS Only Marching Ones.  Base Address = 0006C000.

 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * * * * * * * * *
The time taken for section 32 is 0 minutes and 23 seconds.

-------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 032
CAS Only Marching Ones.  Base Address = 0006C020.

 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * * * * * * * * *
The time taken for section 32 is 0 minutes and 23 seconds.

-------------------------------------------------------------------------------
MA_test card no. = 0.

SECTION 033
Simplified Marching Ones. Base Address = 0006B4C0.

 *
The time taken for section 33 is 0 minutes and 19 seconds.

-------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 033
Simplified Marching Ones. Base Address = 0006B4E0.

 *
The time taken for section 33 is 0 minutes and 18 seconds.
-------------------------------------------------------------------------------
```

```
MA_test card no. = 0.

SECTION 034
Read Hammer Write.

 *
Flushing processor cache.
The time taken for section 34 is 0 minutes and 25 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 034
Read Hammer Write.

 *
Flushing processor cache.
The time taken for section 34 is 0 minutes and 24 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 0.

SECTION 035
Write Hammer Read.

 *
Flushing processor cache.
Checking results.
The time taken for section 35 is 0 minutes and 12 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 035
Write Hammer Read.

 *
Flushing processor cache.
Checking results.
The time taken for section 35 is 0 minutes and 12 seconds.

--------------------------------------------------------------------------------
```

```
MA_test card no. = 0.

SECTION 036
Address Square Wave Test.

 *
The time taken for section 36 is 0 minutes and 3 seconds.
--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 036
Address Square Wave Test.

 *
The time taken for section 36 is 0 minutes and 4 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 0.

SECTION 037
Column Strife Test.

 *
Flushing processor cache.
The time taken for section 37 is 0 minutes and 6 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 037
Column Strife Test.

 *
Flushing processor cache.
The time taken for section 37 is 0 minutes and 6 seconds.

--------------------------------------------------------------------------------
```

```
MA_test card no. = 0.

SECTION 038
Static Soccer Ball Test.

 *
Flushing processor cache.
The time taken for section 38 is 0 minutes and 6 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 038
Static Soccer Ball Test.

 *
Flushing processor cache.
The time taken for section 38 is 0 minutes and 6 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 0.

SECTION 039
Moving Inversion Test.

 *
The time taken for section 39 is 0 minutes and 19 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 039
Moving Inversion Test.

 *
The time taken for section 39 is 0 minutes and 19 seconds.


DIAGNOSTIC COMPLETED
MEM>
```

```
MEM>state

                          STATE VARIABLES
                          ---------------
                     (E=ENABLED   S=SUPPRESSED )

 Error Print      E   Non-Error Print E   Isolation Print E
 Error Pause      S   Non-Error Pause S   Isolation Pause E
 Infinite Looping S   Hardcopy        S

 Section        0000000010    Step              0000000300
 Total loops    0000000001    Loops remaining   0000000000
 Step address     000456F8

 Selected Sections
 0-31    7FE00000    32-63  00000000
 64-95   00000000   96-127  00000000

MEM>reset

MEM>state

                          STATE VARIABLES
                          ---------------
                    (E=ENABLED   S=SUPPRESSED)

 Error Print      E   Non-Error Print E   Isolation Print E
 Error Pause      S   Non-Error Pause E   Isolation Pause E
 Infinite Looping S   Hardcopy        S

 Section        0000000010    Step              0000000300
 Total loops    0000000001    Loops remaining   0000000001
 Step address     000456F8

 Selected Sections
 0-31    80000000    32-63  00000000
 64-95   00000000   96-127  00000000

MEM>exit
```

Example 5: Power Fail Recovery Test

Note

The console path must be switched to 16.0 for this test. Use **conspath**, not **changeio** (**changeio** will not work; it does not change the console path in stable storage or on page 0). Remember to switch it back to the original path after the test is complete.

Caution

This version of the power fail recovery test is destructive. Reboot ISL after exiting this test.

```
ISL>                                          boot ISL
ISL>conspath                                  change console path to 16.0
    Enter system console path:   16.0

ISL> display

    Autoboot is ON (enabled)

    Autosearch is OFF (disabled)

    Primary boot path is 2/4.0.0.0.0.0.0

    Alternate boot path is 6/4.2.3.0.0.0.0

    System console path is 16.0.0.0.0.0.0

ISL>
```

Note

The following steps require a functioning AP card if the **rs** command is used; use of the reset button does not require a functioning AP card.

Next, you must reset the system to copy new stable storage values to page 0. You can do this by pressing the reset button, or by entering a Control-B, which causes the following prompt to appear:

```
CM>                                    AP prompt
CM>rs                                  reset the system using AP card

Execution of this command irrecoverably halts all system processing
and I/O activity and restarts the computer system.

Type Y to confirm your intention to restart the system (Y/N): y
SPU hardware was successfully reset. (APMSG 02)

Processor Dependent Code (PDC) revision 2.0

Console path         = 4.1.0.0.0.0.0
Primary boot path    = 2/4.0.0.0.0.0.0
Alternate boot path = 6/4.2.3.0.0.0.0

96 MB of memory configured and tested.

Autoboot from primary path enabled.
To override, press any key within 10 seconds.

Boot from primary boot path (Y or N)?> n
Boot from alternate path (Y or N)?> y

Interact with IPL (Y or N)?> y

Booting.

Console IO Dependent Code (IODC) revision 1
Boot    IO Dependent Code (IODC) revision 3

Booted.

ISL Revision 2726  June, 1987

ISL> a1100am                       load a1100am memory diagnostic

Possible som_start found at 0003B000.
System_id, a_magic = 020B0107.

Non-sharable executable SOM.
version_id = 05124000.
file_time = 0000000000000000.
entry_space, _subspace,
    _offset = 00000000, 00000000, 000406D0.
aux_header_location = 00000080.
aux_header_size = 00000030.
som_length = 00022800.
aux_header_id = 10000004.
    aux_header_length = 00000028.
HP_UX exc_auxhdr.
```

```
DIAGNOSTIC SYSTEM FOR ISL STAND ALONE DIAGNOSTICS
Version 3.2

Type HELP for command information.
Type INFO for test information.
A1100A SPU Memory Diagnostic
Part Number 30190-90020 VERSION 2.3
Undecoded (raw) Console Path is
    255 255 255 255 255 255 16
Warning: Console Path uses the PDH Direct Port to the AP.
if this message is not displayed, the test will not work!
MEM>section 11/13 select power fail test
MEM>resume


SECTION 011
First part of powerfail test.
Direct port is not configured in both stable storage and page 0.

Switching console path to PDH DIRECT PORT.  Takes about one minute...
        Loading ENTRY_INIT.
        Executing ENTRY_INIT.
        Loading ENTRY_IO.
Console successfully switched to PDH DIRECT PORT.

 *
The time taken for section 11 is 1 minutes and 20 seconds.


SECTION 012
Second part of powerfail test.
    Shut off AC power now!!!!!!!!!!! cycle power
 *
The time taken for section 12 is 0 minutes and 5 seconds.
```

Note

Turn off AC before diagnostic times out (about 30 seconds); otherwise, the test does not work. You will not be prompted to turn the power back on.

Note

The Hex display cycles through the following codes before the program returns, and loops to the beginning of the test section. To exit this routine, enter [Control] (Y), enter [Control] (C), or allow the diagnostic to timeout by NOT responding to any directed actions.

```
CODE: 0000
CODE: 1000
CODE: 1357
CODE: 1357
CODE: 2053
CODE: 2055
CODE: 2058
CODE: CA01

SECTION 013
Soft Error Check.

 *
The time taken for section 13 is 0 minutes and 5 seconds.

DIAGNOSTIC COMPLETED
MEM>
```

Example 6: Running Selected Memory Array Tests

Caution For the array tests which follow, it is mandatory that the corresponding memory controller exist. If it does not, and you attempt to run a test on its arrays, the system will HPMC.

Note If the system displays the message `Diagnostic section incompatible with page zero`, this is an indication that there is not sufficient space remaining to run the test. You can swap arrays, if necessary.

```
MEM>cari 0 8 select memory array cards to be tested
MEM>section 30/39 select memory array tests
MEM>resu        run the diagnostic test sections selected or the default if none selected

HP A1100A SPU Memory Diagnostic
Part Number 31090-90020. Version A.09.02

The HPA of the processor is FFF8A000.

The Hversion = 00000830. The Sversion = 00000413.
PDC_CACHE values are:
I_size = 00080000. I_conf = 01412000. I_base = 00000000.
I_stride = 00000020. I_count = 00002000. I_assoc = 00000002.

SECTION 000

 * Computing CRC on diagnostic SOM.

The time taken for section 0 is less than 1 second.


The IODC of the first MC is 00441B41, 00000900.

First Memory Controller Status:

Reg  9: 00000020 :SPA
Reg 13: 00026440 :Status; DE, ME = 00000000
Reg 14: 00000100 :Control; Enable Error Corr. = 00000100
Reg 16: 00000620 :Error Address
Reg 17: FFFF0097 :Error Syndrome
Reg 32: 07C00000 :IO_MBMASK0
Reg 33: 07C00000 :IO_MBMASK1
Reg 34: 00001000 :IO_MBCONF0
Reg 35: 00000000 :IO_MBCONF1
Reg 36: 80180020 :IO_SPAMASK
Reg 48: 00008000 :Memory Configuration
MA slot #0: 16 Mb board. configuration information
MA slot #1: empty.
MA slot #2: empty.
```

```
MA slot #3: empty.
MA slot #4: empty.
MA slot #5: empty.
MA slot #6: empty.
MA slot #7: empty.
The key values are:
Maximum memory = 33554432. (H) = 02000000.
Maximum SPA = 33554432. (H) = 02000000.
IO_MBMASK0, IOMBMASK1 = 07C00000 : 07C00000
IO_MBCONF0, IO_MBCONF1 = 00001000 : 00000000
IO_SPAMASK = 80180020.
```

```
The IODC of the second MC is 00441B41, 00000900.

Second Memory Controller Status:

Reg  9: 00000020 :SPA
Reg 13: 00026440 :Status; DE, ME = 00000000
Reg 14: 80000100 :Control; Enable Error Corr. = 00000100
Reg 16: 20000620 :Error Address
Reg 17: FFFF0097 :Error Syndrome
Reg 32: 07C00000 :IO_MBMASK0
Reg 33: 07C00000 :IO_MBMASK1
Reg 34: 00001000 :IO_MBCONF0
Reg 35: 00000000 :IO_MBCONF1
Reg 36: 80180020 :IO_SPAMASK
Reg 48: 00008000 :Memory Configuration
MA slot #8: 16 Mb board. configuration information
MA slot #9: empty.
MA slot #10: empty.
MA slot #11: empty.
MA slot #12: empty.
MA slot #13: empty.
MA slot #14: empty.
MA slot #15: empty.
The key values for the second MC are:
Maximum memory = 33554432. (H) = 02000000.
Maximum SPA = 33554432. (H) = 02000000.
```

```
MA_test card no. = 0.

SECTION 030
Data Square Wave test. Base address = 0006B4C0.

 *
The time taken for section 30 is 0 minutes and 1 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 030
Data Square Wave test. Base address = 0006B4E0.

 *
The time taken for section 30 is 0 minutes and 1 seconds.

--------------------------------------------------------------------------------

MA_test card no. = 0.

SECTION 031
Semaphore Wave test.  Base address = 0006B4C0.

 *
The time taken for section 31 is 0 minutes and 1 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 031
Semaphore Wave test.  Base address = 0006B4E0.

 *
The time taken for section 31 is 0 minutes and 1 seconds.
--------------------------------------------------------------------------------
MA_test card no. = 0.

SECTION 032
CAS Only Marching Ones.  Base Address = 0006C000.

 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   * * * * * * * * *
The time taken for section 32 is 0 minutes and 23 seconds.

--------------------------------------------------------------------------------
```

```
MA_test card no. = 8.

SECTION 032
CAS Only Marching Ones.  Base Address = 0006C020.

 *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
  *  *  *  *  *  *  *  *
The time taken for section 32 is 0 minutes and 23 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 0.

SECTION 033
Simplified Marching Ones. Base Address = 0006B4C0.

 *
The time taken for section 33 is 0 minutes and 19 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 033
Simplified Marching Ones. Base Address = 0006B4E0.

 *
The time taken for section 33 is 0 minutes and 18 seconds.
--------------------------------------------------------------------------------
MA_test card no. = 0.

SECTION 034
Read Hammer Write.

 *
Flushing processor cache.
The time taken for section 34 is 0 minutes and 25 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 034
Read Hammer Write.

 *
Flushing processor cache.
The time taken for section 34 is 0 minutes and 24 seconds.

--------------------------------------------------------------------------------
```

```
MA_test card no. = 0.

SECTION 035
Write Hammer Read.

 *
Flushing processor cache.
Checking results.
The time taken for section 35 is 0 minutes and 12 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 035
Write Hammer Read.

 *
Flushing processor cache.
Checking results.
The time taken for section 35 is 0 minutes and 12 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 0.

SECTION 036
Address Square Wave Test.

 *
The time taken for section 36 is 0 minutes and 3 seconds.
--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 036
Address Square Wave Test.

 *
The time taken for section 36 is 0 minutes and 4 seconds.

--------------------------------------------------------------------------------
```

```
MA_test card no. = 0.

SECTION 037
Column Strife Test.

 *
Flushing processor cache.
The time taken for section 37 is 0 minutes and 6 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 037
Column Strife Test.

 *
Flushing processor cache.
The time taken for section 37 is 0 minutes and 6 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 0.

SECTION 038
Static Soccer Ball Test.

 *
Flushing processor cache.
The time taken for section 38 is 0 minutes and 6 seconds.

--------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 038
Static Soccer Ball Test.

 *
Flushing processor cache.
The time taken for section 38 is 0 minutes and 6 seconds.

--------------------------------------------------------------------------------
```

```
MA_test card no. = 0.

SECTION 039
Moving Inversion Test.

 *
The time taken for section 39 is 0 minutes and 19 seconds.

------------------------------------------------------------------------------
MA_test card no. = 8.

SECTION 039
Moving Inversion Test.

 *
The time taken for section 39 is 0 minutes and 19 seconds.


DIAGNOSTIC COMPLETED
MEM>
```

Error Message Information

A1002AM and A1100AM SPU Memory diagnostics generate similar error messages, as described below.

Conventions

The memory diagnostic generally conforms to the following message conventions, except in the cases specified. The following apply to the A1002AM SPU Memory Diagnostic and are similar for the A1100AM SPU Memory Diagnostic. First, the diagnostic prompt is always:

```
MEM>
```

Second, the error messages have a standardized format as follows:

```
ERROR IN SECTION XXX, Step xxx
ERROR at aaaaaaaa. WAS bbbbbbbb. SHOULD BE cccccccc.
```

Where "aaaaaaaa" is the address at which an error was detected. "bbbbbbbb" and "cccccccc" are the actual and expected values respectively.

Exceptions

The exceptions to this convention include:

1. Mask Values

```
ERROR IN SECTION XXX, Step xxx
ERROR at aaaaaaaa. WAS bbbbbbbb. SHOULD BE cccccccc. MASK IS dddddddd.

"dddddddd" has the following values and meanings for section 7:
  0x00000001 - syndrome bit 7      0x00000010 - syndrome bit 3
  0x00000002 - syndrome bit 6      0x00000020 - syndrome bit 2
  0x00000004 - syndrome bit 5      0x00000040 - syndrome bit 1
  0x00000008 - syndrome bit 4      0x00000080 - syndrome bit 0

"dddddddd" has the following values and meanings for section 18:
  0x00000001 - bit 31
  0x00000002 - bit 30
  ....
  0x40000000 - bit 1
  0x80000000 - bit 0
```

2. Device error missing

```
DE=0 at address aaaaaaaa.
```

   Signals that a device error(de) was expected in the status register but none was found.

3. Unexpected device error

```
DE<>0 at address aaaaaaaa.
```

   Signals that a device error(de) was found in the status register but was not expected.

4. Unexpected high priority machine check

```
unexpected HPMC - ECC enabled at address aaaaaaaa.
```

   The test did not expect to get a high priority machine check (HPMC), while the error correction circuitry (ecc) was enabled.

5. Lack of HPMC

```
expected HPMC but didn't get it at address aaaaaaaa.
```

   The test expected a high priority machine check but didn't get one.

6. FRU identification

```
Replace Module xx.
```

   The isolation messages identify which FRU should be replaced. The message specifies which module to replace by printing an indentifying number.

Contents

Tables

5

SPU I/O Diagnostic (A1002AI, A1100AI)

Introduction

The HP A1002AI/A1100AI SPU I/O Diagnostic tests the internal SPU I/O hardware of the Central Processing Unit (CPU), to detect and isolate FRU failures.

Systems Tested by the A1002AI SPU I/O Diagnostic

The following table lists all current system families tested by the A1002AI SPU I/O Diagnostic.

Table 5-1. Systems Tested by A1002AI SPU I/O Diagnostic

Family	Label	Code Name	Models
xx2	xx2	Silverfox	922, 832, 932, etc.
825/925	x25	Firefox	825, 925
	x35	Topgun	835, 935, etc.
	x45	Shogun	845, 949, etc.

Systems Tested by the A1100AI SPU I/O Diagnostic

The following table lists all current system families tested by the A1100AI SPU I/O Diagnostic.

Table 5-2. Systems Tested by A1100AI SPU I/O Diagnostic

Family	Label	Code Name	Models
850/950	x50	Cheetah	850S, 950, etc.
	x55	Jaguar	855S, 955, etc.
	x60	Cougar	960
	x80, x70	Panther	870, 980

Note

The code names listed in the two preceding tables are not official; however you may see them in internal documents or hear them used by factory personnel.

Chapter Organization

This chapter provides the following information about the SPU I/O Diagnostic:

- Functional Description
- Unique Commands
- Test Section Descriptions
- Test Sequence
- Example Session: A1002AI SPU I/O Diagnostic
- Example Session: A1100AI SPU I/O Diagnostic

Defects and Enhancements

Submit defect reports and enhancements requests concerning this diagnostic through the STARS database referencing product number 30342-10003 (A1002AI) and 30342-10001 (A1100AI).

Minimum Configuration

The minimum hardware configuration required to load and run the offline diagnostics system consists of the following functional hardware, firmware, and software items:

- PA-RISC SPU with Main Memory and Processor Dependent Code (PDC)
- Boot device and boot path hardware
- System console
- Initial System Load Code (ISL)
- Offline Diagnostics Software

Note

The A1002AI diagnostic can test a maximum of 2 AFL/CIO Device Adapters or Channel Adapters. These adapters are not necessary for a minimal configuration as far as A1002AI is concerned.

Functional Description

Carefully read and understand this information before attempting to use this diagnostic for the first time. Online help is available to the user with the ISL based Offline Diagnostics Environment "Help" facility.

The SPU I/O Diagnostic tests the following specific hardware (if present and configured) for PA-RISC SPUs.

A1100AI	The test verifies I/O from the VLSI Bus Converter to the internal CIO device adapter within the VLSI Channel adapter. It also tests the BC-> CA-> 6 Port Mux-> S0 path to the AP. These devices make up the minimum I/O hardware configuration.
A1002AI	The test verifies I/O from the VLSI Channel adapter (System card) to the CIO device adapter (CIO card).

Limitations

The tests conducted by the I/O diagnostic are functional only. Intermittent errors require the use of the ISL based Channel Exerciser utility (CAEXR) to reveal stress-related faults.

The diagnostic displays BC and CA configurations, but is not meant to replace the functionality of the ISL based IOMAP utility. This utility is able to locate, identify, and selftest any device attached to the SPUs.

This resolution of the diagnostic varies with the fault source. Best case results isolate a single hardware component such a Bus Converter or Channel Adapter. Worst case results isolate an I/O pathway such as BC<—>CA<—>DA<—>DEVICE.

Remapping (A1100AI SPU I/O Diagnostic only)

The A1100AI diagnostic remaps I/O space as a means of detecting I/O mapping errors. If such an error occurs while the diagnostic executes, the diagnostic will attempt to detect and report the error. However, in this case the integrity of the diagnostic itself cannot be guaranteed, since the an I/O device may map over the memory controller where program memory resides.

Unique Commands

The following command information applies to this diagnostic only. Some examples are Bus Converter dependent while others relate to a specific SPU I/O hardware implementation:

path(A1100AI only)	{SMB fixed field}/{MID_BUS ff}.{HP-CIO slot}.{HP-IB addr.} "path" elements right of MID_BUS fixed field are currently ignored.
	Example = '2' : SMB MODULE 2
	Example = '2/' : BC-X
	Example = '2/4' : MID_BUS-X, SLOT 1
	Example = '2/8' : MID_BUS-X, SLOT 2
	Example = '2/4.1.0' : Same as 2/4.
EVPR	Enables the verbose report.
SVPR	Suppresses the verbose report.
IORE[GISTERS]{ path }	Displays SMB and MID_BUS device registers.
IODC{ path}	Displays SMB and MID_BUS IODC headers. Displays and decodes the first 16 bytes of the module's IODC.
HPAM[AP]	Displays PROCESSOR's, BC's & CA's pertinent I/O address.
FILL[BUFR]{ function}{ pattern} (A1100AI only)	Fills the Write buffer(W_buf) with a sequence of 256 32 bit data patterns.
	{ function} :
	{ ALL0[S]} : Fills buffer with all zeros.
	{ ALL1[S]} : Fills buffer with all ones.
	{ ALLS[AME]}{ pat} : Fills buffer with all one { pat}.
	{ RAND[OM]}{ seed} : Fills buffer with a sequence of pseudo random patterns.
	{ SEQU[ENCE]}{ [pats=n} { pat1,.patn}{ [fill]=pat} :}
COMP[AREBUF] (A1100AI only)	Compares the Read buffer(R_buf) to the Write buffer(W_buf).

Test Section Descriptions

The organization and coverage of the I/O diagnostic is arranged by test section number in ascending (but not necessarily consecutive) order. There are a total of 20 sections. Table 5-1 below lists these test sections according to number and name. The hardware or function tested is stated in the section column for each test.

The test sections are described below. When a test which applies only to a single SPU, that is noted.

Section	Description
1 (A1100A1 only)	**Bus Converter Reset** This test searches MID_BUS for all channel adapters, initializing the structure io_map, tests BC error conditions and calls BC ENTRY_TEST.
2	**Channel Adapter Initialization** This test calls PDC_IODC to load each channel adapter's Entry_init and check the ca status registers. It calls each channel adapter's Entry_init to quick test it, and loads their respective soft physical addresses. If this is the current console path, it is reset for future console activity. Then it checks all status registers.
3	**Channel Adapter Register Test** This test loads each channel adapter's Entry_test, and checks ca status registers. The procedure pointer refers to the local version of Entry_test. The test calls each channel adapters' Entry_test to test their register sets, then checks all status registers. If this is the current console path (A1002A1 only), the path is reset for future console activity.

Table 5-1. I/O Diagnostic Organization and Coverage

Section	Diagnostic	Test Section Name
1	A1100AI	Bus Converter Reset test
2	Both	Channel Adapter initialization test
3	Both	Channel Adapter register test
4	Both	Channel Adapter ram stack test
5	Both	Channel Adapter flex field addressing and DIO loopback test
6	Both	Channel Adapter SRQ test
7	Both	Channel Adapter ARQ test
8	Both	Channel Adapter flex field addressing and DMA loopback test
9	A1100AI	Channel Adapter error status test
10	A1100AI	Terminal Mux Selftest
11	A1100AI	AP Selftest through Terminal Mux (S0) test
12	A1100AI	AP Selftest through PDH Direct Port (DP) test
13	A1100AI	Read HEX DISPLAY from AP, through Terminal Mux
14	A1100AI	Read HEX DISPLAY from AP, through PDH Direct Port
15	A1100AI	AP loopback: DP to console to S0
16	A1100AI	AP loopback: S0 to console to DP
17	A1100AI	Logical Module Selftest
123	A1100AI	SPA WRITE scope loop
124	A1100AI	SPA READ scope loop
125	A1100AI	Direct I/O scope loop
126	A1100AI	DMA scope loop

4	**Channel Adapter ram stack Test** This test loads each channel adapter's Entry_test, and checks ca status registers. The procedure pointer refers to the local version of Entry_test, calls each channel adapter's Entry_test to test its ram stacks, then checks all status registers. If this is the current console path (A1002AI only), the path is reset for future console activity.
5	**Flex field addressing and Direct I/O Loop Back** (Flex field addressing is not done in the A1002AI Diagnostic.) This test loads each channel adapter's Entry_test, and checks ca status registers. If Entry_test is not available from a particular channel, it changes the procedure pointer to a local version of Entry_test, calls each channel adapter's Entry_test to test direct I/O by internal device adapter loop back, then checks all status registers. If this is the current console path, it is reset for future console activity.
6	**Channel Adapter SRQ Test** This test calls PDC_IODC to load each channel adapter's Entry_test, and checks ca status registers. If Entry_test is not available from a particular channel, it changes the procedure pointer to a local version of Entry_test, calls each channel adapter's Entry_test to test its SRQ processing, then checks all status registers. If this is the current console path (A1002AI only), it is reset for future console activity.
7	**Channel Adapter ARQ Test** This test calls PDC_IODC to load each channel adapter's Entry_test, and checks ca status registers. If Entry_test is not available from a particular channel, it changes the procedure pointer to a local version of Entry_test, calls each channel adapter's Entry_test to test its ARQ processing, then checks all status registers. If this is the current console path (A1002AI only), it is reset for future console activity.
8	**Flex field addressing and DMA Loop Back** (Flex field addressing is not done in the A1002AI Diagnostic.) This test calls PDC_IODC to load each channel adapter's Entry_test, and checks ca status registers. If Entry_test is not available from a particular channel, it changes the procedure pointer to a local version of Entry_test, calls each channel adapter's Entry_test to test channel dma by internal device adapter loop back, then checks all status registers. If this is the current console path (A1002AI only), it is reset for future console activity.
9 (AI100AI only)	**Channel Adapter error status test** This diagnostic test first reconfigures the I/O space to standard addresses. Then, each channel adapter's Entry_test is called to test error status handling. After error status handling passes successfully, each status register is checked to ensure that it contains the correct contents.

10 (A1100AI only) **6 Port Mux Selftest** The console path is loaded from stable storage, and Entry_Init for the 6 Port Mux path is loaded and called.

11 (A1100AI only) **AP Selftest from S0** The console path is loaded from stable storage. This is followed by loading Entry_IO for the S0 console path. The Access Port is then commanded to run Selftest by an escape sequence through S0 and read (poll for) Selftest results. Finally, the diagnostic code uses an escape sequence to search for the AP Selftest results contained within the read buffer from S0.

12 (A1100AI only) **AP Selftest from DP** This test causes the AP to run Selftest by escape sequence through DP and read (poll for) the Selftest results. The diagnostic code then uses an escape sequence to search for the AP Selftest results contained within the read buffer from DP.

13 (A1100AI only) **Read front panel by AP escape sequence from S0** This test loads the console path from stable storage and loads Entry_IO for the S0 console path. It causes the AP to send the hex display by escape sequence through port S0 and read (poll for) the hex display. It searches for the AP escape sequence (hex display) within the S0 read buffer and, when found, compares it with the expected hex display.

14 (A1100AI only) **Read front panel by AP escape sequence from DP** This test causes the AP to send the hex display by escape sequence through port DP and read (poll for) the hex display. It searches for the AP escape sequence (hex display) within the DP read buffer and, when found, compares it with the expected hex display.

15 (A1100AI only) **AP loopback - DP to terminal to S0** This test loads the console path from stable storage and loads Entry_IO for the S0 console path. It then writes 1840 characters to the console through Port DP, reads them back through Port S0, and it compares the console block just read with the expected console block.

16 (A1100AI only) **AP loopback - S0 to terminal to DP** This test writes 1840 characters to the console through Port S0, reads them back through Port DP, and compares the console block just read with the expected console block.

17 (A1100AI only)	**Logical Module Selftests** This test loads each channel adapter's Entry_Init in sequential order, and causes invocation of the Selftest for each logical module.
123 (A1100AI only)	**SPA Write (scope loopable)** If UUT 1 has changed, then this test reforms pointers to target HPA and SPA. It writes 32 bits to target SPA page 3, address 14.
124 (A1100AI only)	**SPA Read (scope loopable)** If UUT 1 has changed, this test reforms pointers to target HPA and SPA. It then reads 32 bits to target SPA page 3, address 14.
125 (A1100AI only)	**Channel Adapter Internal DA DIO (scope loopable)** If UUT 1 has changed, this test reforms pointers to target HPA and SPA. It calls local Entry_Test direct I/O internal DA loopback.
126 (A1100AI only)	**Channel Adapter Internal DA DMA (scope loopable)** If UUT 1 has changed, this test reforms pointers to target HPA and SPA. It calls local Entry_Test DMA internal DA loopback.

Test Sequence

To select the Offline I/O Diagnostic, type "A1002AI" or "A1100AI". After loading the diagnostic, type "RESUME" displays the diagnostic banner on the console.

```
HP A1002A/A1035A I/O Diagnostic Version x.x

The HPA of the processor is FFF80000
The processor model is 8.0.4.01.
```

This is followed by a table of I/O configuration and the diagnostic prompt "I/O>". The system then waits for user commands. Commands include selecting sections, isolation pauses, or looping. Hardcopy printout can be selected at this point (A1100AI only). To execute the diagnostic, type "RESUME" again.

The default parameters are:

Parameter	State
Sections (HP A1002A SPU)	1/8
Sections (HP A1100A SPU)	1/126
activity indicators	enabled
error and isolation messages	enabled
pause after error and isolation messages	enabled
looping or hardcopy	no

This diagnostic requires 2 to 3 minutes to execute. See Chapter 3 for command options and syntax.

Run and Resume

If **Resume** has not been entered in this session, typing **Run** provides no information or activity. If **Resume** has been entered already, typing **Run** displays the banner, followed by the I/O display.

```
HP A1002A/A1035A I/O Diagnostic Version x.x :
```

Typing the first **Resume** displays the following.

```
HP A1002A/A1035A I/O Diagnostic Version x.x :
```

(or for the A1100A SPU):

```
HP 3000/950,HP 9000/850 I/O DIAGNOSTIC.  VERSION DATE CODE = 2730
```

A table of all I/O found in the system is then printed, with the following information for each controller.

```
THE FOLLOWING IS A DISPLAY OF THE I/O AS CURRENTLY CONFIGURED.

HPA used in PDC_IODC call = X'xxxxxxxx
: description          -- MID_BUS SLOT=x -- path= x.x.x
```

The diagnostic then displays the current Processor, Bus Converter, and Channel Adapter configuration:

<this example illustrates output for A1002AI>

```
CPU HPA = X'xxxxxxxx

CPU HPA = X'xxxxxxxx : CA SPA = X'xxxxxxxx

The following are the default sections : 2 3 4 5 6 7 8
```

Typing a second **Resume** executes the diagnostic.

Example Session: A1002AI SPU I/O Diagnostic

This example shows a diagnostic session using the A1002AI I/O Diagnostic. The diagnostic is first booted from ISL, then the **help** and **info** commands are entered for display of online command and test section information. This is followed by use of the **state** command to display diagnostic system status information. Next, the **reset** and **resume** commands are entered to set system default values and continue operation of the I/O diagnostic. Entry of the **register** and **exit** commands cause the general register contents to be displayed, followed by termination of the diagnostic programming.

Example 1. Loading the Diagnostic

```
ISL> A1002AI

Beginning of SOM address = 0x00038800
Beginning of code space address = 0x00040000
Last code space address = 0x00059FFC
Last SOM address = 0x00081150
DIAGNOSTIC SYSTEM FOR ISL STAND ALONE DIAGNOSTICS
Version 1.0

Type HELP for command information.

Type INFO for test information.

I/O>help

        Command Keyword Repertoire

 SECTION, LOOP, RESUME, RESET, EEPS, SEPS, EIPS,
 SIPS, ENPS, SNPS, ERRPAUSE, EEPR, SEPR, EIPR,
 SIPR, ENPR, SNPR, ERRONLY, HARCOPY, STATE,
 STOP, LISTIO, CHANGEIO, HELP, INFORMATION,
 DEBUG, DUMP, REGISTER, RUN, EXIT

 enter Help <keyword> for discrete descriptions.
```

Example 2. Obtaining Information and State Data

```
I/O>info

HP A1002A/A1035A I/O Diagnostics Version 1.2
Section 2 Channel Adapter Initialization Section
Section 3 Channel Adapter Register Test Section
Section 4 Channel Adapter Ram Stack Test Section
Section 5 Channel Adapter and Direct I/O Loopback
Section 6 Channel Adapter SRQ Test Section
Section 7 Channel Adapter ARQ Test Section
Section 8 Channel Adapter DMA Loopback

Additional Commands:

EVPR Enable Verbose Print

SVPR Suppress Verbose Print

IORE Display a set of I/O system

IODC Display & decode the 1st 16 bytes of module's IODC


I/O>state

                          STATE VARIABLES
                          ---------------
                      (E=ENABLED   S=SUPPRESSED )

 Error Print      E   Non-Error Print E   Isolation Print S
 Error Pause      E   Non-Error Pause S   Isolation Pause S
 Infinite Looping S   Hardcopy        S

 Section        0000000000    Step              0000000000
 Total loops    0000000001    Loops remaining   0000000000
 Step address     00000000

 Selected Sections
 0-31    3F800000    32-63  00000000
 64-95   00000000   96-127  00000000
```

Example 3. Using Reset and Examine State

```
I/O>reset;resume;state

Section 002
000 001 002 003 004 005 006 007

Section 003
...
.
.
.

Section 008
000 001 002 003 004 005

                          STATE VARIABLES
                          ---------------
                     (E=ENABLED   S=SUPPRESSED )

 Error Print      E   Non-Error Print E   Isolation Print E
 Error Pause      S   Non-Error Pause E   Isolation Pause E
 Infinite Looping S   Hardcopy        S

 Section        0000000000    Step              0000000000
 Total loops    0000000001    Loops remaining   0000000001
 Step address     00000000

 Selected Sections
 0-31    3F800000    32-63  00000000
 64-95   00000000   96-127  00000000
```

Example 4. Examine Registers and Exit

```
I/O>register


--------------------------------------------------------------------------------
General Registers : sampled within : Section=008, Step=002
--------------------------------------------------------------------------------
r00 / 00000000 00084800 0004E0E4 00000880  00018000 000406D0 1F0010DC 00007401
r08 / 0000000B 0000000D 00001000 00008000  00000CB8 00000000 FFFFFFFF 00000000
r16 / 00000080 1F0010DC 1F0010F8 000000FF  00019B0B 00000001 00000000 00000001
r24 / FFF7E000 00000000 00064B20 0005D000  00000000 FFF7F034 000199C8 00100010
--------------------------------------------------------------------------------
I/O>exit

Exiting.....
```

Example 5. Running Test Sections

The following illustrates screen output while a diagnostic is executing.

```
I/O>resume

Section 002
000 001 002 003 004 005 006 007

Section 003
000 001 002 003 004 005

Section 004
000 001 002 003 004 005

I/O>exit

Exiting.....
```

Example Session: A1100AI SPU I/O Diagnostic

The following dialog illustrates the use of the **info**, **state**, and **resume** commands.

Example 1. Obtaining Information

```
I/O>info


HP_3000/950, HP9000/850 I/O DIAGNOSTIC.  VERSION 1.0

ADDITIONAL USER COMMANDS SPECIFIC TO THIS DIAGNOSTIC:

        'path' is {SMB fixed field}/{MID_BUS ff}.{HP-CIO slot}.{HP-IB addr.}
        'path' elements right of MID_BUS fixed field are currently ingored.
               Example = '2'    : SMB MODULE 2
               Example = '2/'   : BC-0
               Example = '2/4'  : MID_BUS-0, SLOT 1
               Example = '2/8'  : MID_BUS-0, SLOT 2
               Example = '2/4.1.0'      : Same as 2/4.

        'I/O>EVPR' Enables the verbose report.
        'I/O>SVPR' Enables the terse report.
        'I/O>IORE[GISTERS]{ path}' Displays SMB and MID_BUS device registers.
        'I/O>IODC{ path}' Displays SMB and MID_BUS IODC headers.
        'I/O>HPAM[AP]' Displays PROCESSOR's, BC's & CA's pertinent I/O addrs.
        'I/O>FILL[BUFR]{ function}{ patterns}'
               Fills the Write buffer(W_buf) with a sequence of 256 32 bit
               data patterns.
                { function} :
                { ALLO[S]} : Fills buffer with all zeros.
                { ALL1[S]} : Fills buffer with all ones.
                { ALLS[SAME]}{ pat} : Fills buffer with all one { pat}.
                { RAND[OM]}{ seed} : Fills buffer with a sequence of pseudo
                  random patterns.
                { SEQU[ENCE]}{ [pats]=n}{ pat1,.patn}{ [fill]=pat} :}
                  Fills buffer with a sequence of user selected patterns.
        'I/O>COMP[AREBUF]' Compares the Read buffer(R_buf) to the Write
                  buffer(W_buf).
```

```
THE FOLLOWING IS A DESCRIPTION OF THE TEST SUITE:
        SECTION_01: Bus Converter initialization tests.
        SECTION_02: Channel Adapter initialization tests.
        SECTION_03: Channel Adapter register tests.
        SECTION_04: Channel Adapter ram stack tests.
        SECTION_05: Channel Adapter flex field addressing and DIO loopback.
        SECTION_06: Channel Adapter SRQ tests.
        SECTION_07: Channel Adapter ARQ tests.
        SECTION_08: Channel Adapter flex field addressing and DMA loopback.
        SECTION_09: Channel Adapter error status test.
        SECTION_10: Terminal Mux Selftest.
        SECTION_11: AP Selftest through Terminal Mux (SO).
        SECTION_12: AP Selftest through PDH Direct Port (DP).
        SECTION_13: Read HEX DISPLAY from AP, through Terminal Mux.
        SECTION_14: Read HEX DISPLAY from AP, through PDH Direct Port.
        SECTION_15: AP loopback: DP to console to SO.
        SECTION_16: AP loopback: SO to console to DP.
        SECTION_17: Logical Module Selftest.
        SECTION_123: SPA WRITE scope loop.
        SECTION_124: SPA READ scope loop.
        SECTION_125: Direct I/O scope loop.
        SECTION_126: DMA scope loop.
```

Example 2. Using The State Command

```
I/O>state


                          STATE VARIABLES
                          ---------------
                      (E=ENABLED   S=SUPPRESSED )

 Error Print       E   Non-Error Print E   Isolation Print S
 Error Pause       E   Non-Error Pause S   Isolation Pause S
 Infinite Looping S    Hardcopy        S

 Section         0000000000    Step              0000000000
 Total loops     0000000001    Loops remaining   0000000000
 Step address      00000000

 Selected Sections
 0-31    7FFFC000    32-63  00000000
 64-95   00000000   96-127  00000000
```

Example 3. Running Test Sections

```
I/O>section 1/10

I/O>state



                          STATE VARIABLES
                          ---------------
                      (E=ENABLED   S=SUPPRESSED )

 Error Print      E   Non-Error Print E   Isolation Print S
 Error Pause      E   Non-Error Pause S   Isolation Pause S
 Infinite Looping S   Hardcopy        S

 Section        0000000000    Step              0000000000
 Total loops    0000000001    Loops remaining   0000000000
 Step address     00000000

 Selected Sections
 0-31    7FE00000    32-63  00000000
 64-95   00000000   96-127  0000001E


I/O>seps
```

Example 4. Using the Resume Command

```
I/O>resume


HP_3000/950, HP 9000/850 I/O DIAGNOSTIC.  VERSION 1.0

THE FOLLOWING IS A DISPLAY OF THE I/O AS CURRENTLY CONFIGURED.


HPA used in PDC_IODC call  = X'FFF80000
 : SMB Memory Controller. -- MC-0 -- path= 0.x.x

HPA used in PDC_IODC call  = X'FFF82000
 : Bus Converter: SMB-MID_BUS. -- BC-0 -- path= 2/x.x.x
---------------------HPA's assigned to MID_BUS.----------------------------

HPA used in PDC_IODC call  = X'FFF44000
 : VLSI HP-CIO Channel Adapter. -- MID_BUS-0 SLOT=1 -- path= 2/4.x.x

HPA used in PDC_IODC call  = X'FFF48000
 : VLSI HP-CIO Channel Adapter. -- MID_BUS-0 SLOT=2 -- path= 2/8.x.x
--------------------------------------------------------------------------------

HPA used in PDC_IODC call  = X'FFF86000
 : Bus Converter: SMB-MID_BUS. -- BC-1 -- path= 6/x.x.x
---------------------HPA's assigned to MID_BUS.----------------------------
BUS CONVERTER IS NOT CURRENTLY CONFIGURED.
--------------------------------------------------------------------------------


HPA used in PDC_IODC call  = X'FFF8C000
 : A1100A Processor. -- PROCESSOR 3 -- path= 12.x.x

HPA used in PDC_IODC call  = X'FFF90000
 : A1100A PDH-AP(Console DP). -- PDH DP -- path= 16.x.x
```

The following displays the current Processor, Bus Converter, and Channel Adapter configuration:

```
PROCESSOR HPA = X'FFF8C000


BC HPA = X'FFF82000 : IO_IO_LO = X'FFF00000 : IO_IO_HI = X'FFF80000

        CA HPA = X'FFF44000 : CA SPA = X'FFF3E000

        CA HPA = X'FFF48000 : CA SPA = X'FFF3C000


BC HPA = X'FFF86000 : IO_IO_LO = X'FFE80000 : IO_IO_HI = X'FFF00000

SWITCHING CONSOLE PATH TO PDH DIRECT PORT.....
        Loading ENTRY_INIT.
        Executing ENTRY_INIT.
        Loading ENTRY_IO.
CONSOLE SUCCESSFULLY SWITCHED TO PDH DIRECT PORT.

Selected Sections : 1 2 3 4 5 6 7 8 9 10 123 124 125 126

PAUSE BEFORE EXECUTING SELECTED SECTIONS....
I/O>resume



SECTION 0001
This section resets, configures and tests all Bus Converters.
000 001 002 003 004 005 006 007 008 009 010 011 012 013
I/O SYSTEM IS OFF!!! <The BC/DA I/O path is disabled during testing.>


SECTION 0002
This section configures all Channel Adapters and resets them and
all Logical Modules.
000 001 002 003 004 005 006 007 008 ... 012 013 014 015 016 017 018 019
THE CONSOLE MUX PATH IS OFF!!! <The BC/DA console path is disabled.>

SECTION 0003
This section tests channel adapter registers.
000 001 002 003 004 005

SECTION 0004
This section tests all channel adapters ram stack.
000 001 002 003 004 005

SECTION 0005
This section tests internal loop-back on each channel adapter with direct I/O.
```

Error Message Information

The I/O diagnostic generally conforms to the following message conventions.

- The diagnostic prompt is always: I/O>.
- The error messages have a standard format:

Note

Error messages for this diagnostic are embedded in the program code, so there is not a specific list of error messages. The diagnostic responds to the specific error encountered when generating an error message. The error messages given below show the type of information commonly provided.

Error Message Format

Error messages conform the format given below.

```
ERROR IN SECTION xxx, STEP xxx
The following error message is returned from ENTRY_TEST:
xxxxxxxxx

        HPA/SPA = X'xxxxxxxx/X'xxxxxxxx
```

Error Message Examples

The following examples illustrate messages generated by this diagnostic.

Message 1 - Power Transient on Channel Adapter

```
ERROR IN SECTION 002, STEP 012
CHANNEL ADAPTER PWR TRANSIENT AFTER PDC_IODC : HPA = X'FFEC4000, path=6.4.x

        CHANNEL ADAPTER REGISTERS : HPA = X'FFEC4000 :
         -- MID_BUS-1 SLOT=1 -- path= 6/4.x.x
   IO_EIM.........FFFFFFFF  IO_DC_DATA.....00004F06  IO_SPA_ADDRESS.00000000
   IO_STATUS.......FD010040=( dbyte dend br myad PEHI PELO lost ry )
   INTERRUPT.......00000000 CHAIN_RAM_BASE.FFFFFFFF  SUB_MASK_CLR....00000000
   SUB_MASK_SET....00000000 DIAGNOSTIC.....00000000  NMI_ADDRESS.....FFFFFFFF
   NMI_DATA........FFFFFFFF


ERROR IN SECTION 002, STEP 014
CHANNEL ADAPTER PWR TRANSIENT AFTER PDC_IODC : HPA = X'FFEC8000, path=6.8.x

        CHANNEL ADAPTER REGISTERS : HPA = X'FFEC8000 :
         -- MID_BUS-1 SLOT=2 -- path= 6/8.x.x
   IO_EIM.........FFFFFFFF  IO_DC_DATA.....00008F06  IO_SPA_ADDRESS.00000000
   IO_STATUS.......FD010040=( dbyte dend br myad PEHI PELO lost ry )
   INTERRUPT.......00000000 CHAIN_RAM_BASE.FFFFFFFF  SUB_MASK_CLR....00000000
   SUB_MASK_SET....00000000 DIAGNOSTIC.....00000000  NMI_ADDRESS.....FFFFFFFF
   NMI_DATA........FFFFFFFF
```

Message 2 - Error at Hex Display

```
ERROR IN SECTION 013, STEP 003
        AP DELIVERS UNEXPECTED FRONT PANEL HEX DISPLAY!!
        EXPECTED FRONT PANEL VALUE      = X'CE80
        AP DELIVERED FRONT PANEL VALUE  = X'000&3^

ERROR IN SECTION 014, STEP 001
        AP DELIVERS UNEXPECTED FRONT PANEL HEX DISPLAY!!
        EXPECTED FRONT PANEL VALUE      = X'CE80
        AP DELIVERED FRONT PANEL VALUE  = X'0000
```

Message 3 - Error at Flex Broadcast

```
ERROR IN SECTION 001, STEP 009
BUS CONVERTER HARD ERROR ON SMB AFTER BROADCAST FLEX : HPA = X'FFF8F000 ...

BUS CONVERTER REGISTERS :
SMB-REGISTERS : HPA = X'FFF8F000:
---------------
        IO_STATUS =             X'00000140 he ry
        IO_ERROR_ADDRESS =      X'F1040020
        IO_ERROR_MASTER =       X'02008000
        IO_IO_LO =              X'FFE80000
        IO_IO_HI =              X'FFFF0000

MID_BUS REGISTERS : HPA = X'FFEC0000 : -- MID-BUS-1 SLOT=0 -- path = 15/0.x.x
        THIS BUS CONVERTER IS OFF!!
```

Message 4 - Error After Reset

```
ERROR IN SECTION 001, STEP 001
BUS CONVERTER ERROR : HPA = X'FFF82000 , path= 2/x.x.x
IO_IO_LO - IO_IO_HI range of addresses remain valid after CMD_RESET.

BUS CONVERTER REGISTERS :
SMB-REGISTERS : HPA = X'FFF82000:
---------------
        IO_STATUS =             X'00000040 ry
        IO_ERROR_ADDRESS =      X'FFF82040
        IO_ERROR_MASTER =       X'02008000
        IO_IO_LO =              X'F1000000
        IO_IO_HI =              X'FFF80000

MID_BUS-REGISTERS : HPA = X'FFF40000 :  -- MID_BUS-0 SLOT=0 -- path= 2/0.x.x
---------------
        IO_STATUS =             X'00000040 ry
        IO_ERROR_ADDRESS =      X'00012001
        IO_ERROR_MASTER =       X'4400C000
000
```

Contents

6

Input/Output Map Utility (IOMAP)

Introduction

This chapter describes the IOMAP Utility and provides operating procedures for it. IOMAP displays the configuration of all devices attached to an HP Precision Architecture RISC SPU and the modules and adapters connected to it. This utility runs on both the MPE XL and HP-UX variants of these SPUs. IOMAP provides identification, selftest, and loopback tests on each component capable of such tests. IOMAP can perform the following:

- Identify the configuration of I/O components in the system.
- Quickly test all I/O components in a system.
- Test selected individual components.
- Determine the configuration of Co-processors, Analyzer Card, Cache and TLB memory sizes, ROM revisions, and switch settings.
- Display version data from board revision code EEPROMS (850/950 family only).

This chapter provides the following information about IOMAP:

- Functional Description
- User Interface Description
- Error Message Information

Defects and Enhancements

Submit defect reports and enhancement requests concerning this diagnostic through the STARS database, referencing product number 30345-10001 (IOMAP).

Minimum Configuration

The minimum hardware necessary for IOMAP to load and run consists of:

- Any HP Precision Architecture RISC computer
- Functioning boot path
- Front panel hex/LED display
- Console (required for 850/950 family SPU)

Functional Description

The IOMAP utility resides in the ISL boot directory. It can only be called from ISL, when the operating system is not running. The system is completely unavailable to the customer while IOMAP is running. IOMAP can be loaded from either a supported boot device (cold-load) tape drive, or disk drive (start device).

Most of IOMAP is written in the C programming language. A small portion of the low level code is written in assembler. Mnemonics and abbreviations used are listed in the glossary at the end of this manual.

IOMAP provides four test modes:

- **Identify** This test attempts to identify each component in every I/O path (or specified paths). The information printed includes I/O path to the component, component name, component ID number, component software model number(if applicable), firmware revision (if applicable), hardware revision (if applicable), and an indication of which test modes are available for the component. Configuration data is determined by PDC calls. Path information is obtained from PDC calls, direct I/O (DIO), and DMA transactions.
- **Loopback** This test performs component dependent loopback tests where feasible. The result of this test is reported as a pass, fail, unimplemented or untestable status.
- **Selftest** This test initiates the internal selftest of each component where feasible. The result of this test is reported as a pass, fail, unimplemented or untestable .
- **View** This test examines the version code of each board on the system (850/950 family only). The display appears on the console only, not the hex display. When running IOMAP in *silent* mode, no output appears.

Default Test Sequence

The default IOMAP test consists of the following:

- Display the current configuration of the processor, including the presence of Co-processor boards and Analyzer Cards, the memory sizes of Cache and TLB cards, the processor model number, and the PDC firmware revision.
- Check all possible I/O paths to determine the components present and identify them: module, bus converter, adapter, device or unit. IOMAP then displays a table showing all configured components.

Selftest or loopback diagnostic tests can be specified for all testable components. When specified, it is performed after mapping. Error messages are printed for any component that fails a test. The user may also limit the identify, selftest, loopback and view tests to specified path.

Limitations on Selftest and Loopback

IOMAP performs selftest and/or loopback tests on all components with those capabilities. IOMAP currently cannot test the following:

- Boards lacking these features (such as Channel Adapter or Parallel Card).
- Hewlett Packard Precision Bus (HP-PB) modules on 808/815 family (such as the serial controller) which are not part of the console or boot path.
- Devices attached to ports of the terminal multiplexor (such as terminals, printers and datacomm lines).

Special Test Requirements

Adhere to the HP-PB device adapter test requirements given below.

HP-PB LAN Device Adapter

Before running selftest and loopback, terminate the external link. Attach a terminated T-connector to the LAN connector of the MAU. Connect it to the currently configured MAU (internal or external).

HP-PB GPIO Device Adapter

Attach a loopback test hood before running the external loopback test.

User Interface

The user interface is divided into input and output sections. Each of these sections can operate in different modes, as specified by the user. To obtain online help for **IOMAP** enter "iomap help" at the ISL> prompt.

User Input

The user input interface provides a means of configuring **IOMAP**, allowing selective user control over **IOMAP** operation. Configurations and options can be specified at the **IOMAP** command line. This provides a fast means for an experienced user to set up **IOMAP**, and also allows operation to be preconfigured in an autoboot file. An interactive interface is also available which prompts the user for specific input and provides configuration instructions.

While running **IOMAP**, a user can enter "Break Mode", which suspends program operation. The Break Mode "debug" facility permits trained support personnel to examine and modify status and registers.

Caution

While Break Mode can allow the user to harmlessly display **IOMAP** internal variables, it also invokes a powerful debug facility. If inadvertently used it may hang the SPU and cause an HPMC.

Commands and Syntax

The default run command is shown below:

```
ISL>iomap <optional parameters/keywords>
```

The default test settings for the commands are:

Command	State
debug	not enabled
defaults	no (see note below)
errcount	infinite (0)
erronly	false
help	false
loop	once
noerrpause	false
path	all
silent	false
tests	identify only

Note

The defaults listed above are *only* enabled if 'defaults' is entered on the command line. If one or more commands (other than `defaults`) are input to configure specific settings, the remaining settings take on default values. If no parameter commands are entered, `IOMAP` automatically invokes the interactive mode.

The parameter keywords and syntax are given below. Supplying parameters causes **IOMAP** to bypass the interactive interface. If parameters are entered on the command line, **IOMAP** accepts them and sets the other parameters to the defaults.

Using the command line input to bypass the interactive dialog does not affect the printing of status or error messages on the console. To limit console output to error messages only (status and error messages appear on the hex/LED display), specify "erronly". To limit message output to the hex/LED display only, specify "silent".

IOMAP parameter keywords are listed below. The optional part of the keyword appears inside the square brackets, "[ ]," with the required part to the left. Several keywords are followed immediately by a required equals sign, "=". Optional and required parameters are as shown for each keyword. All numeric input is in decimal. All periods "." and commas "," are required if shown, unless they are contained in square brackets "[ ]" and everything within those brackets is omitted.

deb[ug]

Print a trace of program execution while IOMAP is running. It requires source code to understand; drives the hex/LED display and, if not in silent mode, prints on the console.

The default is debug not enabled.

def[aults]

Specifies that default values are to be used. The defaults for each keyword are shown in this section. Use of this keyword causes the interactive interface to be bypassed.

errc[ount]

Instructs IOMAP to abandon execution and return to ISL after the specified number of errors are detected. The *nn* parameter is the number of errors in decimal. The '=' is required. The default is to allow infinitely many errors without returning to ISL. Note that entering the value 0 (zero) allows an infinite number of errors.

```
err[count]= nn
```

erro[nly]

Causes IOMAP to print only error messages. It inhibits all informational messages about running status. Erronly is overridden by silent mode, which guarantees that no error or informational messages are printed after the configuration dialog. The default is erronly false. All informational messages are normally printed.

h[elp]

Prints information about **IOMAP** on the console before returning to ISL. It gives a brief explanation of the purpose of **IOMAP**, and enough syntax to prepare the user for the non-interactive interface. The default is help false, meaning that the help messages are not normally printed.

l[oop]

Specifies the number of times that **IOMAP** should repeat the selected tests. The '=' is required; *nn* is decimal. The default is for **IOMAP** to perform each selected test once before returning to ISL. Note that entering the value 0 (zero) allows an infinite number of loops.

```
l[oop]= nn
```

n[oerrpause]

Instructs **IOMAP** to not suspend execution when an error occurs. A hex/LED code is displayed, a message is printed on the console (if not inhibited by silent mode), and then execution continues.

The default is noerrpause false, meaning that **IOMAP** execution normally suspends for user intervention if an error occurs.

p[ath]

Restricts **IOMAP**'s operation to a specified I/O path. **IOMAP** only performs its tests (identify, selftest, loopback and view) on components in the specified I/O path. The '=' is required.

The default is for **IOMAP** to perform specified tests on all existing I/O paths. Use the form p[ath]= n.n ... n For example, to run tests on the HP-IB card in CIO slot 2 of module 8, the command line would be `iomap path=8.2`

FOR HP INTERNAL USE ONLY

r[evport]:

Instructs **IOMAP** to display the revision information of the board specified (850/950 family only). It is used in conjunction with t[est]=v[iew]. The '=' is required; *nn* is decimal. Valid revision port numbers are from 1 to 27.

Default is to display revision information of all boards (*nn* =0).

```
r[evport]=nn
```

s[ilent]

Prevents **IOMAP** from printing anything to the console after the interactive dialog. All communications from **IOMAP** to the user are with the hex/LED display.

If parameters are specified on the command line, and silent mode is used, **IOMAP** does not print anything on the console at any point in the program. No console or console path is required in this case. The default is silent false.

t[ests]

Specifies which tests are to be performed. The four tests are identify, selftest, loopback and view. View applies to the 850/950 family only. View and the other three tests are mutually exclusive. If any other test is selected, view will not be run. The "all" option may be used to specify the first three tests. The '=' is required.

The default is identify only.

```
t[ests]=[i[dentify][,s[elftest]][,l[oopback]]]
 or
t[ests]=a[ll]
 or
t[ests]=v[iew]
```

v[iew] (850/950 family only)

Displays the version code of each board on the system. This appears on the console only, not on the Hex display. When running **IOMAP** in silent mode, no output appears.

Break Mode

The user can break the program at certain points by using [Control] (C) or [Control] (Y). These user interrupts are detected after each loop completes, and after any input or error message completes.

Note

Break mode is not entered by using the console "break" key. Enter [Control] (C) several times to ensure successful execution.

Caution

The read and write commands represent a true debug facility, which can easily destroy the state of the machine and cause a High Priority Machine Check. These debug commands should only be used by someone with a detailed knowledge of IOMAP and system internals. With explicit instructions (followed exactly), these commands are useful for examining the state of the DMA data buffers.

The following message is printed when the user presses [Control] (C) or [Control] (Y) on the keyboard. It is the first indication that break mode has been entered:

```
BREAK MODE [c,d,e,h,r,s,w] (c):
```

Break mode commands are given below:

Command	Description
c	Continue - resume program execution
d	Display - displays the current configuration values.
e	Exit - terminate IOMAP and return to ISL
h	Help - print a menu of break mode commands, with a brief explanation of the purpose of each.

r	Read - reads real memory and displays the contents on the console.
	`Syntax: r <addr in hex/LED>`
	`[,<number of words to read in hex/LED>]`

s	Status - show the detailed status information for each configured device adapter.
w	Write - writes the supplied pattern to real memory at the specified address. `Syntax:w <addr in hex/LED>,<data in hex/LED>[,<number of words to write in hex/LED>`
q	same as 'e'
?	same as 'h'

Diagnostic Output

IOMAP provides status and error information to the user through the front panel hex/LED display or the console. Selecting "silent" mode routes error/status information only to the hex/LED display. Otherwise, status and error messages are output to the console in parallel with hex/LED display output.

Hex/LED Display Format (Silent Mode Only)

Hex codes for IOMAP fall within the range of CE80 - CEBF, using defined and undefined codes, as required by the HP Precision Architecture RISC Chassis I/O standard. Since this range is not sufficient to express all of the required information, the hex/LED codes are followed by descriptive "parameter" values. The "class" code (CE80 - CEBF) is displayed first, followed by the stated number of "parameter" values.

For 840 and 850/950 family SPUs, the hex display shows the "class code" for three seconds, then displays the parameter values for two seconds each, as shown below.

```
CEBA  n  n  n
 |    |  |  |
 |    |  |  type ID
 |    |  |
 |    |  module, slot or device slot number
 |    |
 |    level in the I/O tree
 |
 Initial Code
```

For any system with only four LEDs (xx2 and 825/935 families), the LED displays cycle in the following sequence:

Data	Interval
blank	3 seconds
hex digit 1	2 seconds
blank	1 second
hex digit 2	2 seconds
blank	1 second
hex digit 3	2 seconds
blank	1 second
hex digit 4	2 seconds
blank	1 second
[parameter 1]	2 seconds

(more blanks and parameters, as required)

The cycle requires at least 15 seconds, depending on the number of parameters passed by the message.

For the 808/815 SPUs, the LED display cycles in the following sequence:

Data	Interval
blank	3 seconds
hex digits 1 & 2	2 seconds
blank	1 second
hex digit 3 & 4	2 seconds
blank	1 second
[parameter 1]	2 seconds

(more blanks and parameters, as required)

The cycle requires at least 9 seconds, depending on the number of parameters passed by the message.

Note

For the 808/815, xx2, and 825/925 family SPUs, the only truly readable message information appears on the console. Disabling "silent" mode speeds up program execution by eliminating pausing. This feature speeds up access to error information whenever the console is available.

Hex error codes are defined in the "Error Message Information" listing under the "Output to Hex Display and Console" category.

Hex/LED Display Output

When `IOMAP` execution begins, the display shows the running code "CE81". As each test executes, the current loop count appears. If an error occurs, the error described in the previous section appear.

If `IOMAP` is in silent mode, the map data appears as follows:

1. The code CEBA appears in the hex/LED display.
2. A number indicating the depth of the component in the I/O tree appears:

   (up to 6 levels of bus converters are permitted),

   device adapter
3. The component module, device or unit number appears.
4. The type ID for the component appears.

Any errors are displayed when detected and then the program continues.

For example, if the following is the console display for a 840 system using default IOMAP settings,

```
I/O Configuration:

                                                    Type SW   Revisions Tests
Path          Component Name                         ID  Mod  Hdwr Firm Avail
------------ ---------------------------------- ---- ---- ---- ---- -----
8             CIO Channel Adapter                   8H  10H   0    0
8.0               HP-IB card                        2H   -    0    0 ST LB
8.0.3                 7914 Disc Drive             20bH   -    0    0
8.0.3.0                  Hard Disc Unit             0H   -    0    0
8.0.3.1                  Cartridge Tape             0H   -    0    0
8.2               HP-IB card                        2H   -    0    0 ST LB
8.2.0                 2608A Dot Matrix Printer    001H   -    0    0
8.2.1                 7970E Mag Tape Controller   183H   -    0    0
```

then the following would be displayed in the hex display.

```
CEBA 1 8 8
CEBA 7 0 2
CEBA 8 3 20B
CEBA 9 0 0
CEBA 9 1 0
CEBA 7 2 2
CEBA 8 0 2001
CEBA 8 1 183
CEBA 7 3 3
```

If the following is a typical display for 850/950 family SPUs,

```
I/O Configuration:

                                                    Type SW   Revisions Tests
Path         Component Name                          ID  Mod  Hdwr Firm Avail
------------ ---------------------------------- ---- ---- ---- ---- -----
2            Bus Converter                          7H   CH    0    0
2/8            CIO Channel Adapter                  8H  14H    0    0
2/8.0              HP-IB card                       2H    -    0    0 ST LB
2/8.0.3              7914 Disc Drive               20bH    -    0    0
2/8.0.3.0                 Hard Disc Unit            0H    -    0    0
2/8.0.3.1                 Cartridge Tape            0H    -    0    0
2/8.2              HP-IB card                       2H    -    0    0 ST LB
2/8.2.0              2608A Dot Matrix Printer     2001H    -    0    0 ST LB
2/8.2.1              7970E Mag Tape Controller     183H    -    0    0
```

then the following would be displayed in the hex display ...

```
CEBA 1 2 7
CEBA 2 8 8
CEBA 7 0 2
CEBA 8 3 20B
CEBA 9 0 0
CEBA 9 1 0
CEBA 7 2 2
CEBA 8 0 2001
CEBA 8 1 183
```

If the following is the normal display for 808/815 family SPU,

```
I/O Configuration:
                                          Type      SW Revisions Tests
Path         Component Name                ID     Mod Hdwr Firm Avail
------------ ------------------------  ----   ---- ---- ---- -----
4            HP-PB HP-IB Card                      4H  40H    0    0 ST LB
4.1             7933H/7935H disc drive           212H    -    -    -
5            Serial Controller                     5H   EH    0    0 ST LB
                System Console is connected to this module
6            HP-HIL                                5H  14H    0    0 ST LB
8            Memory Controller                     1H   9H    0    0    LB
28           Memory Controller                     1H   8H    0    0    LB
```

then the following would be displayed in the hex/LED display ...

```
CEBA 1 4  4
CEBA 8 1  212
CEBA 1 5  5
CEBA 1 6  5
CEBA 1 8  1
CEBA 1 28 1
```

Any errors would be displayed when detected and then the I/O mapping would continue.

The four or eight LEDs will display the same hex codes in sequence.

Console Messages

The console provides information in addition to the hex/LED display. In some cases, the console messages provide more information than the hex/LED display.

IOMAP can be run without a terminal by specifying "silent mode". Disabling "silent" mode speeds up the console display. When "silent" mode is enabled, **IOMAP** always outputs error and status information to the hex/LED display.

Silent mode allows test execution without a console, or whenever console I/O may be interrupted or corrupted. Use of silent mode is discouraged, since it is slow and more difficult to use.

Example Session

This example session illustrates a typical session with console output.

Example 1. 840 Session

The following console output illustrates IOMAP functions and displays on a 840 SPU. The interface is indentical on the 825/925 family and 808/815 family SPUs. Examples 2 and 3 show portions of the display specific to the 825/925 family and 808/815 family SPU.

```
ISL> iomap

IOMAP Revision A.01.00  February 21, 1989

IOMAP Running: CE81

This program has the capability to identify the configuration of
the system and its I/O paths and devices.  Many of the components of
the I/O system can be tested with selftest and loopback diagnostics.

Without changing any parameters, this program will map all existing
I/O components in the system, but will not perform any other diagnostics.

You may enter break mode by pressing control-C or control-Y.

You may press control-X to erase what you just typed in.


Do you wish to modify any program parameters? [y,n] (n):  y

Change tests to be executed? [y,n] (n):  y

    Identify? [y,n] (y):  y

    Selftest? [y,n] (n):  y

    Loopback? [y,n] (n):  y

Test single path only? [y,n] (n):  n

Enter the number of LOOPS [<n>,0=infinite] (1):

Change miscellaneous program parameters? [y,n] (n):  y

    Pause on errors? [y,n] (y):

    Maximum error count before returning to ISL (infinite):

    Suppress all messages? [y,n] (n):

    Print error messages only? [y,n] (n):
```

```
    Display software debug messages? [y,n] (n):


Identify: Loop 1:  All I/O components are being identified...

Processor Identification:
    Hardware Model:  4H (9740A), Revision:  0
    Hardware ID:  0000102dH, Software ID:  0000102dH
    Processor Dependent Code (PDC) Revisions:
        SM - PDC Firmware:  6      IU - PDC Selftest:  1
    Processor Board Revisions:
        SM - System Monitor:    0
        RF - Register File:     0
        EU - Execution Unit:    1
        IU - Instruction Unit:  0
        TL - TLB Board:         1
        CA - Cache Unit:        6
    Cache and TLB Sizes:
        Instruction Cache:  64 K bytes, Instruction TLB:  2 K entries
        Data Cache:         64 K bytes, Data TLB:         2 K entries
    Co-processors:
        Floating Point Co-processor is installed
    Main Memory:  24 M bytes

I/O Configuration:
                                                    Type   SW Revisions Tests
Path         Component Name                          ID   Mod Hdwr Firm Avail
------------ ---------------------------------- ---- ---- ---- ---- -----
8            CIO Channel Adapter                     8H  10H    0    0
8.0              HP-IB card                          2H    -    0 2612 ST LB
8.0.0                7933/7935 disc drive          212H    -    -    -
8.0.1                7933/7935 disc drive          212H    -    -    -
8.0.2                9144 cartridge tape drive     260H    -    -    -
8.1              Console Device Adapter
8.2              HP-IB card                          2H    -    0 2612 ST LB
8.2.1                2608A dot matrix printer     2001H    -    -    - ST LB
8.2.3                7978A mag tape unit           178H    -    -    -
8.4              LAN card                            6H    -    1 2620 ST
8.6              A-LINK card                         8H    -    8 2812 ST LB
8.6.0                7937FL disc drive                -    -    -
8.6.1                7937FL disc drive                -    -    -
12           Bus Converter (Local Port)              7H   CH    0    0 ST LB
12/0             Bus Converter (Remote Port)         7H   CH    0    0 ST LB
12/24            HP-PB MUX Card                      5H   DH    0    0 ST LB
24           CIO Channel Adapter                     8H  10H    0    0
24.2             HP-IB card                          2H    -    2 2613 ST LB
24.3             HP-IB card                          2H    -    2 2613 ST LB
36           Memory Controller ( 8 M Bytes)          1H   8H    0    0    LB
44           Memory Controller ( 8 M Bytes)          1H   8H    0    0    LB
52           Memory Controller ( 8 M Bytes)          1H   8H    0    0    LB
```

```
Identify Loop 1 (1H) complete.

Selftest: Loop 1: All testable I/O components are being tested...
Processor Selftest:  Untestable
Component    Component Name                                Selftest Results
------------ --------------------------------------------  ----------------
8            CIO Channel Adapter                           Untestable
8.0              HP-IB card                                3 Sec ST Passed
8.0.0                7933/7935 disc drive                  Unimplemented
8.0.1                7933/7935 disc drive                  Unimplemented
8.0.2                9144 cartridge tape drive             Unimplemented
8.1              Console Device Adapter
8.2              HP-IB card                                3 Sec ST Passed
8.2.1                2608A dot matrix printer              3 Sec ST Passed
8.2.3                7978A mag tape unit                   Unimplemented
8.4              LAN card                                  3 Sec ST Passed
8.6              A-LINK card                               3 Sec ST Passed
8.6.0                7937FL disc drive                     Unimplemented
8.6.1                7937FL disc drive                     Unimplemented
12           Bus Converter (Local Port)                    Passed
12/0             Bus Converter (Remote Port)               Passed
12/24            HP-PB MUX Card                            Passed
24           CIO Channel Adapter                           Untestable
24.2             HP-IB card                                3 Sec ST Passed
24.3             HP-IB card                                3 Sec ST Passed
36           Memory Controller ( 8 M Bytes)                Untestable
44           Memory Controller ( 8 M Bytes)                Untestable
52           Memory Controller ( 8 M Bytes)                Untestable

Selftest Loop 1 (1H) complete.

Loopback: Loop 1: All testable I/O components are being tested...

Component    Component Name                                Loopback Results
------------ --------------------------------------------  ----------------
8            CIO Channel Adapter                           Unimplemented
8.0              HP-IB card                                64 Bytes Passed
8.0.0                7933/7935 disc drive                  Unimplemented
8.0.1                7933/7935 disc drive                  Unimplemented
8.0.2                9144 cartridge tape drive             Unimplemented
8.1              Console Device Adapter
8.2              HP-IB card                                64 Bytes Passed
8.2.1                2608A dot matrix printer              64 Bytes Passed
8.2.3                7978 mag tape unit                    Unimplemented
8.4              LAN card                                  Unimplemented
8.6              A-LINK card                               256 Bytes Passed
8.6.0                7937FL disc drive                     Unimplemented
8.6.1                7937FL disc drive                     Unimplemented
12           Bus Converter (Local Port)                    Passed
12/0             Bus Converter (Remote Port)               Passed
```

```
12/24           HP-PB MUX Card                              Passed
24          CIO Channel Adapter                             Unimplemented
24.2            HP-IB card                                  64 Bytes Passed
24.3            HP-IB card                                  64 Bytes Passed
36          Memory Controller ( 8 M Bytes)                  Passed
44          Memory Controller ( 8 M Bytes)                  Passed
52          Memory Controller ( 8 M Bytes)                  Passed

Loopback Loop 1 (1H) complete.

Do you want to exit this program and return to ISL? [y,n] (n):  n

Do you wish to modify any program parameters? [y,n] (n):  y

Change tests to be executed? [y,n] (n): y

    Identify? [y,n] (y): n

    Selftest? [y,n] (n): y

    Loopback? [y,n] (n): y

Test single path only? [y,n] (n):  y

    Enter I/O path <n.n...n>: 8.4

Enter the number of LOOPS [n,0=infinite] (1):

Change miscellaneous program parameters? [y,n] (n):

Selftest:  Loop 1:  Only the component at 8.4 is being tested...
Component    Component Name                                 Selftest Results
------------ ---------------------------------------------  ----------------
8            CIO Channel Adapter                            Untestable
8.4              LAN card                                   15 Sec ST Passed

Selftest Loop 1 (1H) complete.


Loopback:  Loop 1:  Only the component at 8.4 is being tested...

Component    Component Name                                 Loopback Results
------------ ---------------------------------------------  ----------------
8            CIO Channel Adapter                            Unimplemented
8.4              LAN card                                   Unimplemented

Loopback Loop 1 (1H) complete.

Do you want to exit this program and return to ISL? [y,n] (n):  y

IOMAP Exiting.
```

Example 2. Processor Identification Display (825/925 family)

The 825/925 family display resembles the 840 except for the processor identification. The display appears as follows:

```
Processor Identification:
    Hardware Model:  8H (A1002A), Revision:  0
    Hardware ID:  0000303bH, Software ID:  0000303bH
    Processor Board Revisions:
      CPU - CPU Chip:                     2
      SIU - System Interface Unit:        1
      CCU - Cache Control Unit:           2
      TCU - TLB Control Unit:             2
      MIU - Math Interface Unit:          2
      PDH - Processor Dependent Hardware: 1
      PDC - Processor Dependent Code:     1
    Cache and TLB Sizes:
        Instruction Cache:  16 K bytes, Instruction TLB:  1 K entries
        Data Cache:         16 K bytes, Data TLB:         1 K entries
    Co-processors:
        Floating Point Co-processor is installed
    Main Memory:  32 M bytes
```

Example 3. Processor Identification Display (808/815 family)

The 808/815 family display resembles 840 except for the processor identification. The display appears as follows:

```
Processor Identification:
    Hardware Model:  100H (A1408A), Revision:  0
    Hardware ID:  00000000H, Software ID:  00000000H
    Processor Board Revisions:
         ROM Component Revision:       0
    Cache and TLB Sizes:
        Instruction Cache:  256 bytes, Instruction TLB:  32 entries
        Data Cache:           0 bytes, Data TLB:         32 entries
    Co-processors:
        Floating Point Co-processor is installed
    Main Memory:  4 M bytes
```

Note

If a LAN card fails selftest or loopback test, the error code returned from the test will be displayed immediately below the "*** FAILED ***" message. To determine what the error is and what causes it, please run the Local Area Network Device Adapter Diagnostic (LANDAD) within the On-Line Diagnostic System.

The following message is printed after an error occurs, to prompt the user for input:

```
An error was detected; do you want to continue? [y,n] (y):
```

The program suspends until the user responds affirmatively. The *noerrpause* mode inhibits this message and the suspend condition. A response of *n* will disable prompting on errors associated with the current loop and cause the program to exit; a response of *y* will continue program execution.

Example 4. Processor and I/O Displays (850/950 family)

The 850/950 family display differs from the 840 since it shows bus converter paths and can display board revision levels. The processor identification appears as follows:

```
ISL> iomap

IOMAP Revision A.01.00  February 21, 1989

IOMAP Running: CE81

This program has the capability to identify the configuration of
the system and its I/O paths and devices.  Many of the components of
the I/O system can be tested with selftest and loopback diagnostics.

The program can also be used to examine the revision information of each
board (revision port) on the A1100A system.

Without changing any parameters, this program will map all existing
I/O components in the system, but will not perform any other diagnostics.

You may enter break mode by pressing control-C or control-Y.

You may press control-X to erase what you just typed in.


Do you wish to modify any program parameters? [y,n] (n):  y

Do you want to read revision port information? [y,n] (n): y

Which revport? [d=done,0=all,<1..27>] (0): 1

Slot: 1  Board: PDH Board

Addr   Information       Bits   Data (Hex unless otherwise indicates)
0x00   Assembly Number   00-39  3019060002 (BCD)
0x05   Slot Number       00-07  1
0x06   Date Code         00-15  2752 (BCD)
0x08   Hardware ID       00-31  0000000000 (BCD)
0x0C   SMR Serial Number 00-47  CCCCCC (ASCII)
0x12   Division Number   00-15  47 (ASCII)
0x14   Boot ID           00-31  00000000
0x18   Software ID       00-31  00000000
0x1C   Reserved for PDH  00-31  00000000

Next revport? [y,n] (y): n

Which revport? [d=done,0=all,<1..27>] (0): 9

Slot: 9  Board: Mem Array #7
```

```
Slot selected is empty.

Next revport? [y,n] (y): n

Which revport? [d=done,0=all,<1..27>] (0): 27

Slot: 27  Board: BC-Y (SMB slot #3)

Addr   Information       Bits   Data (Hex unless otherwise indicates)
0x00   Assembly Number   00-39  3019060030 (BCD)
0x05   Slot Number       00-07  1B
0x06   Date Code         00-15  2752 (BCD)
0x08   Hardware ID       00-31  0000000000 (BCD)
0x0C   SMR Serial Number 00-47  CCCCCC (ASCII)
0x12   Division Number   00-15  47 (ASCII)
0x14   Boot ID           00-31  00000000
0x18   Software ID       00-31  00000000
0x1C   Reserved for PDH  00-31  00000000

End of revports.

Which revport? [d=done,0=all,<1..27>] (0): d

Do you want to exit this program and return to ISL? [y,n] (n): y

IOMAP Exiting.
```

Hex/LED Display Output Interface

Hex codes are displayed in the front panel hex/LED display at different times during the execution of `IOMAP`. Hex code must be within the range of CE80 - CEBF, using defined and undefined codes, to meet the specifications in the Chassis I/O Architecture document. This is not sufficient to express all of the desired information, so they are followed by descriptive *parameter* values. The *class* code (CE80 - CEBF) is displayed for three seconds, followed by the stated number of *parameter* values displayed for two seconds each.

This section gives the text and meaning of each error message. Each error message is represented by a pair of hex codes. The first one is error class number and the second one is the error code within the error class. There are five classes of errors: input, Mid-bus, HP-PB, DA and other (execution) errors. Within each classes, there are different error codes. Because different machine has different hardware configuration, not all error messages are applicable to all machines. For example, the 815 does not have channel adapters. Therefore, any reference to channel adapter in the error messages will not apply to 815. In this section, "[]" is used to place machine dependent display. For example, bus converter number is placed in "[]" indicating that it is displayed on a system that has a bus converter. Each console message has a corresponding output to the hex/LED display.

Input Error CE90

HEX Error ce90 cea0

CONSOLE *** Expected '=' after "*kkk*" keyword

CAUSE An equals sign '=' must follow immediately after the ***kkk*** keyword. Keyword ***kkk*** represents one of the following command line keywords: ***errcount***, ***loop***, ***path***, or ***tests***.

HEX Error ce90 cea1

CONSOLE *** Expected number after "*kkk*="

CAUSE A decimal number must follow after the equals sign '=', following keyword ***kkk*** on the command line. Keyword ***kkk*** represents one of the following command line keywords: ***errcount***, ***loop***, ***path***, or ***tests***.

HEX Error ce90 cea2

CONSOLE *** Unrecognized keyword "*kkk*"

CAUSE This keyword, which was entered on the command line interface, does not match a valid keyword.

HEX Error ce90 cea3 <*ASCII value of 'c'*>?

CONSOLE *** Expected 'c' as delimiter

CAUSE The command line interface expected *c* as a delimiter, but another character was supplied.

HEX Error ce90 cea4

CONSOLE *** Syntax error

CAUSE This error can occur in many places. It means that the user data entered was not in a correct or a recognizable form. Correct syntax is given in prompt.

HEX Error ce90 cea5

CONSOLE *** Number cannot be negative

CAUSE The loop count and error count cannot be entered as negative numbers.

HEX Error ce90 cea6 [bc] ca da

CONSOLE *** [bc/]ca.da are reserved for the console

CAUSE Any attempt to configure console will cause this message.

HEX Error ce90 cea7 *n*

CONSOLE *** Invalid module number *n*

CAUSE This means that a Precision Architecture (Spectrum) module number outside the range 1 - 63 was entered. *n* is the module number supplied by the user.

HEX Error ce90 cea8 [bc] n

CONSOLE *** Module [bc/] n does not exist

CAUSE This means that a Spectrum (PA) module number is valid but the corresponding module is not installed in the system.

HEX Error ce90 cea9 [bc/] ca

CONSOLE *** Mid_bus module [bc/] ca is not a Channel Adapter

CAUSE The Mid_bus module number, *ca*, is valid but corresponds to a module other than a channel adapter.

HEX `Error ce90 ceaa n`

CONSOLE `*** Invalid Device Adapter number n`

CAUSE The Device Adapter number entered by the user was not between 0 and 15, inclusive.

HEX `Error ce90 ceab`

CONSOLE `*** Number out of range`

CAUSE The number entered is too large or too small to be accepted. The value must be within the range specified by the prompt.

HEX `Error ce90 ceac [bc] ca da`

CONSOLE `*** On [BC bc,] CA ca, Device Adapter da does not exist`

CAUSE The specified Device Adapter number does not correspond with a DA installed on the specified Channel Adapter.

HEX `Error ce90 cead [bc] ca da`

CONSOLE `*** On [BC bc], CA ca, Device Adapter da is not an HP-IB DA`

CAUSE The Device Adapter specified does not correspond to an HP-IB DA. The number may correspond to another type of device adapter.

HEX `Error ce90 ceae`

CONSOLE `*** Unrecognized TESTS keyword` *kkk*

CAUSE This keyword entered after *tests=* is not *all* or a comma separated list of *identify*, *selftest*, or *loopback*. This error occurs in the command line interface only.

HEX `Error ce90 ceaf`

CONSOLE `*** Revport information is not available on this machine.`

CAUSE The machine does not have Revision Code EEPROMS and therefore the information is not available.

HEX `Error ce90 ceb0`

CONSOLE `*** No tests specified`

CAUSE At least one test mode (Identify, Selftest, or Loopback) must be specified for **IOMAP** to perform any meaningful testing.

HEX `Error ce90 ceb1 n[/n]/n`

CONSOLE `*** Module n[/n]/n does not exist`

CAUSE This means that the module number is valid but the corresponding module is not installed in the system.

HEX `Error ce90 ceb2 n`

CONSOLE `*** Module n does not exist`

CAUSE This means that the SMB module number is valid but the corresponding module is not installed in the system. This error message applies to CHEETAH only.

HEX `Error ce90 ceb3 n`

CONSOLE `*** Module n does not exist`

CAUSE This means that the VSC module number is valid but the corresponding module is not installed in the system.

HEX	`Error ce90 ceb4 n n`
CONSOLE	`*** Module n/n does not exist`
CAUSE	This means that the VSC adapter slot is valid but the corresponding slot is empty.

HEX	`Error ce90 ceb5 n n n`
CONSOLE	`*** Module n/n/n does not exist`
CAUSE	This means that the VSC adapter module is valid but the corresponding module is not installed in the system.

Mid-bus Error CE93

HEX	`Error ce93 cea0 [bc] ca stat`
CONSOLE	`*** On [BC bc], Channel Adapter ca module error: Status = stat`
CAUSE	The specified Channel Adapter has had a module error. The contents of the module's status register is displayed as *stat*.

HEX	`Error ce93 cea1 [bc] ca stat`
CONSOLE	`*** [On BC bc, ] Channel Adapter ca timeout: Status = stat`
CAUSE	The CA io_stat register ca_ready bit did not become set (ready) with a reasonable interval (currently one second). This prevented a DMA transfer from being started by software.

HEX	`Error ce93 cea2 mod stat`
CONSOLE	`*** PDC_IODC failed while reading ENTRY_TEST: Module = mod, Status = stat`
CAUSE	An error was returned from the PDC call PDC_IODC while attempting to read ENTRY_TEST IODC from the card at the specified module.

HEX	`Error ce93 cea3 mod stat`
CONSOLE	`*** PDC_IODC failed while reading ENTRY_INIT: Module = mod, Status = stat`
CAUSE	An error was returned from the PDC call PDC_IODC while attempting to read ENTRY_INIT IODC from the card at the specified module.

HEX	`Error ce93 cea4 mod stat`
CONSOLE	`*** Error executing ENTRY_INIT on Module mod: Status = stat`
CAUSE	An error occurred while attempting to execute the ENTRY_INIT IODC on the specified module.

HEX	`Error ce93 cea5 mod`
CONSOLE	`*** SCSI card not 'Ready for Command': Module = mod`
CAUSE	SCSI card at module mod is reporting that it is not ready for a command. This error is usually a result of the SCSI card not accepting the Inquiry command.

HEX	`Error ce93 cea6 mod`
CONSOLE	`*** Unexpected Bus Service Request on SCSI card at Module mod`
CAUSE	This error is a result of a SCSI device unexpectantly asserting the Bus Service Request line during the Inquiry command.

HEX	`Error ce93 cea7 mod`
CONSOLE	`*** Inquiry command to SCSI card at Module mod did not complete`
CAUSE	SCSI card at module mod was unable to complete the Inquiry command.

HEX	`Error ce93 cea8 mod stat`
CONSOLE	`*** Fatal error returned from SCSI card at Module mod: Status =` *stat*
CAUSE	SCSI card at module mod reported a fatal error. Most likely the result of being unable to execute a SCSI Inquiry command.

HEX	`Error ce93 cea9 mod stat`
CONSOLE	`*** Bad status returned from SCSI card at Module mod: Status =` *stat*
CAUSE	SCSI card at module mod returned a bad status value of stat. This most likely occurred while attempting to execute a SCSI Inquiry command.

DA Error CE95

HEX	`Error ce95 cea0 [bc] ca da`
CONSOLE	`*** Device Adapter 'SLOW' switch (S1-7) must be down`
CAUSE	The Device Adapter speed switch S1-7 must be 'down'(toward the board surface). DIP switch S1 is at the front edge of the board.

HEX	`Error ce95 cea1 [bc] ca da`
CONSOLE	`*** Device Adapter 'SCTL' switch (S1-6) must be up`
CAUSE	The 'SCTL' (system controller) switch must be set 'up' (away from the board surface). 'SCTL' is the DIP switch on the board front edge.

HEX	`Error ce95 cea2 [bc] ca da`
CONSOLE	`*** Device Adapter load resistor pack is missing`
CAUSE	the resistor pack must be in place for the HP-IB card to operate at the maximum speed.

HEX	`Error ce95 cea3 [bc] ca da`
CONSOLE	`*** On [BC bc,] CA ca, Device Adapter da failed selftest`
CAUSE	The DA Sense Register PST bit is not set, indicating that the DA selftest failed last time it was run.

HEX	`Error ce95 cea4 [bc] ca da`
CONSOLE	`*** On [BC bc,] CA ca, Device Adapter da is not 'Ready For Command'`
CAUSE	The DA Sense Register RFC bit is not set, meaning that the DA is not able to accept another command. It should be ready at this time.

HEX `Error ce95 cea5 [bc] ca da`

CONSOLE `*** On [BC bc,] CA ca, Device Adapter da timeout error`

CAUSE A DMA transfer initiated for this DA failed to complete within a reasonable time period.

HEX `Error ce95 cea6 [bc] ca da stat`

CONSOLE `*** On [BC bc,] CA ca, Device Adapter da status error: Status = stat`

CAUSE The Device Adapter status does not match the expected value, after a DMA transfer was performed.

HEX `Error ce95 cea7 [bc] ca n stat`

CONSOLE `*** On [BC bc,] CA ca, subchannel n not ready: Status = stat`

CAUSE On the CA, this subchannel's status register subc_ready bit did not come ready within a reasonable period of time (currently one second). This prevented a DMA transfer from being started by software.

HEX `Error ce95 cea8 [bc] ca da stat`

CONSOLE `*** On [BC bc,] CA ca, Subchannel da ARQ timeout: Status = stat`

CAUSE The one second timeout expired before ARQ was seen in the DA sense register.

HEX `Error ce95 cea9 [bc] ca da stat`

CONSOLE `*** On [BC bc,] CA ca, Subchannel da was not destroyed: Status = stat`

CAUSE After ARQ was seen in the DA sense register, the attempt to Destroy SubChannel on the Device Adapter by Direct I/O failed (meaning that no SubChannel Destroyed message was received back from the DA).

HEX `Error ce95 ceaa da`

CONSOLE `*** Unable to reset SCSI bus on Device Adapter da`

CAUSE An attempt to reset the SCSI bus on device adapter da failed. The SCSI bus is reset before the first Inquiry command is issued.

HEX `Error ce95 ceab da`

CONSOLE `*** Inquiry command to SCSI card on Device Adapter da did not complete`

CAUSE An Inquiry command to the SCSI card at device adapter da failed to complete without error.

Other Execution Error CE96

HEX `Error ce96 cea0 index stat`

CONSOLE `*** Error in call to Processor Dependent Code (PDC): Index = index , Status = stat`

CAUSE A call to PDC routine with index index returned a bad status. It is possible that printing this line may not be successful, since output to the console requires repeated PDC calls.

HEX `Error ce96 cea1`

CONSOLE `*** Fatal error--internal buffer(s) overwritten`

CAUSE `IOMAP` has suffered an internal data management problem. If it is not a bad copy of `IOMAP`, then there may be a software bug or hardware problem.

HEX `Error ce96 cea2 n`

CONSOLE `*** Maximum error count (n) exceeded; program will abort.`

CAUSE The maximum error count specified n has been exceeded. Errors associated with the current loop will be displayed and then the program will return to ISL.

HEX `Error ce96 cea7`

CONSOLE `*** Component failed selftest`

CAUSE The current component (last one displayed) failed selftest.

HEX `Error ce96 cea8`

CONSOLE `*** Component failed loopback test`

CAUSE The current component (last one displayed) failed loopback.

Contents

7

Channel Exerciser Utility (CAEXR)

Introduction

The Channel Exerciser Utility (**CAEXR**) is a diagnostic that exercises the 19744A Channel Adapter (CA) board set or equivalent hardware in the HP 9740A, A1002A, or A1100A SPU, including VLSI versions of the CA. This chapter contains the following information:

- Functional Description
- Interface Description
- Example Sessions
- Error Message Information

This diagnostic tests normal Channel Adapter functions while running the CA at maximum attainable DMA speed. It tests the Channel Adapter for any abnormal DMA completion status, and examines the transferred data for any corruption.

CAEXR uses HP-IB Device Adapter cards on the CIO bus of each Channel Adapter under test to generate DMA activity. On the A1100A, **CAEXR** exercises the bus converters. Using the current Channel Adapter and HP-IB Device Adapters, the maximum data throughput has been measured at 2.6 Mbytes/second. Since the DA cards are capable of 1.0 - 1.33 Mbytes/sec, three HP-IB DAs performing concurrent DMA transfers saturate the CIO bus.

Saturation testing of the CIO bus helps locate arbitration problems that are difficult to find in other ways. The HP-IB cables are tested for integrity at speed. **CAEXR** also exercises the CPU cache and Memory Controller modules, periodically concentrating activity in each region of memory.

Defects and Enhancements

Submit defect reports and enhancement requests through the STARS database, referencing 30345-10002 (CAEXR).

Minimum Configuration

CAEXR requires the following hardware and software:

- Any PA-RISC SPU with Main Memory and PDC. Memory requirements are: 3 Mbyte (09740A), 8 Mbyte (A1002A), 16 Mbyte (A1100A)
- Functioning boot path
- Front panel Hex/LED display
- System Console
- One Channel Adapter
- 3 HP-IB cards (2 is minimum but may not saturate the CA)
- Bus Converter (A1100A only)

All error and status messages can be routed to the front panel hex display (HP 9740A/A1100A SPU) or LED display (A1002A SPU).

A minimum CAEXR test requires one BC (A1100A only), one CA and two HP-IB DA cards. To thoroughly test a CA, three HP-IB DAs should be used. This can include a spare HP-IB card from a CE kit.

CAEXR normally operates up to three Channel Adapters and nine HP-IB Device Adapters in a maximum configuration on 9740A and A1002A and up to 12 CAs and 36 DAs on A1100A. It is possible to test up to 12 busy DAs and 12 pair DA's on 9740A. On A1002A, since the 3 CAs are limited to 8 slots each, the maximum possible test configuration is 12 pair DAs. An interactive configuration dialog is available. Error messages can be printed on the console as well as being displayed on the hex/LED display.

The minimum memory required to run the test is one controller with 3 Mb of memory in 9740A, 8 Mb of memory in A1002A and 16 Mb of memory in A1100A. CAEXR is capable of interacting with an unlimited amount of memory, testing the CAs ability to talk with any memory which could be installed. Supported hardware configurations, however, limit this to 128Mb.

Functional Description

Carefully read and understand this information before attempting to use CAEXR for the first time. CAEXR also contains a help facility which is invoked by entering **caexr help** on the command line input.

CAEXR is a utility program which is located on the Support Tape or system disk. It can only be called from ISL, when the system is not running the OS. The system is completely unavailable to the customer while CAEXR is running. CAEXR can be loaded from either a supported boot device (cold-load) tape drive, or a supported boot device (start device) disk drive. Most of CAEXR is written in the C programming language. A small portion of the low level code is written in assembler.

DMA activity in CAEXR uses HP-IB Device Adapter cards on the CIO bus of each Channel Adapter under test. A portion of the DMA transfers are generated by two HP-IB DA cards, which have their HP-IB ports tied together in a loopback configuration. The balance of the DMA activity is generated by a single HP-IB DA card, writing data to the "bit bucket" through its HP-IB cable. This configuration saturates the CIO bus while using the minimum amount of hardware. Saturation occurs at ahout 2.6 Mbytes/sec.

The A1100A SPU provides the added complication of either one or two bus converters that must be accounted for when configuring the test environment. Finally, CAEXR does exercise the bus converter(s) present in the A1100A SPU.

Physical Hardware Configuration

To set up the hardware to run CAEXR, proceed as follows:

1. Power off the system.
2. Connect the DA pairs. The pair(s) of HP-IB Device Adapters form a loopback link.
3. Boot to ISL and load CAEXR.
4. Answer the configuration dialog for the configuration under test. The "BUSY" DAs are plugged into the bus, but are not connected to another DA or device. For minimum systems, disconnect the HP-IB cable that links the tape drive to the system and define that HP-IB DA as the "BUSY" DA.

   1 Mbyte is often sufficient.
5. Begin test execution.
6. When the test completes, reconnect the tape drive HP-IB cable. If the tape drive is disconnected, the system will be unable to regain the ISL> prompt and will hang.

Test Sequence

The following is a description of each pass of CAEXR. It describes the main loop of CAEXR, with 2 or 3 HP-IB DAs configured on one CA.

CAEXR provides instructions for connecting HP-IB cables between the device adapter pairs. See the screen dumps later in this section.

Note

It is possible to configure and operate up to 36 DAs distributed on up to 12 CAs. This large configuration requires twelve times as many DMA transfers to be set up and tested, twelve Moving Buffer addresses and twelve Target Buffers.

The Primary DA is initially set up to be the Master or Controller In Charge (CIC) of the HP-IB, and the Secondary DA is set up to be Slave (not the CIC). These roles may be switched periodically, as described in step 12, below.

1. Software generates the address for the Moving Buffer, and writes it into the channel I/O quad program for the Slave DA in a pair.
2. Software initializes each word of the Source Buffer and the Reference Buffer to semi-random data (both buffers contain the same data). Each word of the Target Buffer is initialized to a fixed data pattern, C5693A96 Hex. Each word of the Moving Buffer is initialized to B5AE2D8A Hex.
3. If a Busy DA is configured on this Channel Adapter, software starts a long DMA write from the memory area under test to the HP-IB (and into the bit bucket). This guarantees that the CIO bus will be saturated when the transfers described below occur.
4. In the first DMA transaction of the transfer, the Master DA writes while the Slave DA reads. The Master DA writes data from the Source Buffer in main memory to the HP-IB cable. The Slave DA reads from the HP-IB, and writes this data to main memory in the Moving Buffer.
5. After the first DMA transaction has completed, the Master DA reads and the Slave DA writes while performing the second transaction. The Slave DA writes the data in the Moving Buffer from main memory to the HP-IB. The Master DA reads this data from the HP-IB and into the Target Buffer.

6. While the DMA transactions are occurring, the CPU generates memory accesses in the memory area under test. This is implemented as reads to avoid interfering with the DMA transfers.
7. After the DA pair DMA transactions have completed, software checks all participating DAs and the CA for the correct status. The link words in the link status lists should point to the correct quads, indicating that each transaction completed. The pair DAs and the busy DA should all have correct completion status. If any discrepancy is seen, a detailed summary of errors is printed on the terminal and/or shown in the front panel hex/LED display.
8. The Reference, Source, Moving and Target buffers are word-wise compared for any differences. If any discrepancies are seen, the word values of each buffer at that address are displayed on the terminal. A buffer compare error status is displayed on the hex/LED display.
9. If any errors were found in steps (7) or (8), the program pauses, unless the NOERRPAUSE option had been selected by the user. It waits for user intervention before retrying the transfer and continuing operation.
10. After 10, 15 or 30 transfers, (30 / number of modes), the program begins to test the next area of memory. Note that a transfer consists of steps 1-9. Testing walks through memory until the end of highest configured memory, then starts over with testing of the lowest configured memory area. The memory area under test is 256 Kbytes long, and each pair DMA transaction is 32 Kbytes.
11. When testing starts over in low memory, the next mode is selected if more than one mode is configured. The modes are: Subchannel, Logchannel and Block logchannel.
12. If the NOSWAP option is false, after each configured mode has been selected once, the Primary and Secondary DA in each pair exchange the Master/Slave roles. If the exchange occurs, steps 1-11 are repeated once.
13. The loop counter is incremented. If the optional loop count has not been exceeded or if no loop count was specified, the next loop starts. This means that the program repeats steps 1-12.

The user interface is divided into input and output sections. Each of these sections can operate in different modes, as specified by the user.

The user output interface is how CAEXR provides status and error information to the user. The front panel hex/LED display on the SPU always outputs this type of information. If the "silent" mode is selected by the user, then the hex/LED display is the only source of information from CAEXR to the user. Otherwise, status and error messages are output to the console in parallel with hex/LED display output.

User Input

The input interface provides a means of configuring CAEXR, allowing selective user control over CAEXR operation. Configurations and options which are entered by command line input can be specified when CAEXR is invoked. This provides a fast means for an experienced user to set up CAEXR, and also allows operation to be preconfigured in an autoboot file.

A more friendly user input interface is optionally available. This interactive interface prompts the user for each piece of information and provides specific configuration instructions. A subset of this interface can be selected with the "expert" mode.

While CAEXR is running, a user can enter "Break Mode" suspending program operation. Break Mode can allow the user to harmlessly display CAEXR internal variables or can invoke a powerful debug facility which, if inadvertently used may hang the SPU and cause an HPMC. This facility is intended to be used only by support personnel who are familiar with detailed program structure of CAEXR and system internals.

Commands and Syntax

The default run command is shown below:

```
ISL> caexr  <optional parameters/keywords>
```

Default command values for CAEXR are listed below:

FOR HP INTERNAL USE ONLY

Parameter	State
busy	8.3 16.3 24.3
debug	not enabled
defaults	false (meaning like IOMAP?)
errcount	infinite (0)
erronly	false
expert	false
help	false
loop	indefinitely (0)
memory	3MB and up
mpx	all
noerrpause	false
noswap	false
pair	8.0, 8.2; 16.0,16.2; 24.0,24.2
silent	false

The keywords and syntax of these parameters are given below. One purpose of the command line input is to bypass all or part of the interactive configuration dialog. If any configuration value parameter is entered on the command line, CAEXR uses all of the supplied values, with the balance of the values being set to their defaults. If the "expert" parameter is supplied by the user, then the interactive user interface is reduced to a terse dialog which is suitable for an experienced user.

If any keyword is specified on the command line (except "expert"), then the interactive user interface is bypassed. If no keyword is specified, or "expert" (only) is specified, then the interactive user interface is invoked.

Using the command line input to bypass or reduce the interactive configuration dialog does not affect the printing of status or error messages on the console. To limit console output to error messages only (status and error messages are still presented on the hex/LED display), the "erronly" mode must be specified on the command line. To limit message output to the hex/LED display only, "silent" mode can be specified.

Note

The autoboot file on any boot medium can be changed to run CAEXR automatically when the system boots from that device. All configuration parameters can be preconfigured by using the command line input feature. "Silent" mode can be specified to keep messages from being printed to the console. This means that CAEXR can be run without a console or console path.

The following text explains the **CAEXR** command line input syntax. Each keyword is listed below. The optional part of the keyword is shown inside the square brackets, "[ ]," with the required part to the left.

Several keywords are followed immediately by a required equals sign, "=". Optional and required parameters are as shown for each keyword. All numeric input is in decimal. All periods "." and commas "," are required if shown, unless they are contained in square brackets "[ ]" and everything within those brackets is omitted.

b[usy]

Precedes the configuration data for all Busy DAs to be configured into **CAEXR**. The "=" and "." are required. Each BC number (<BusyxBC>) is the decimal module # on the SMB; each CA number (<BusyxCA>) is the decimal module #; each DA number (<BusyxDA>) is the decimal slot #; each Device address (<BusyxDV>) is the HP-IB address of the device connected to the DA.

09740A and A1002A defaults for eacb Busy DA are: 8.3 16.3 24.3

A1100A defaults for each Busy DA are: 2/4.3 2/8.3 8/4.3 8/8.3

```
b[usy]=[&<Busy1BC/>]&<Busy1CA>.&<Busy1DA>[.&<Busy1DV>]
       [&<Busy2BC/>]&<Busy2CA>.&<Busy2DA>[.&<Busy2DV>]
       [&<Busy3BC/>]&<Busy3CA>.&<Busy3DA>[.&<Busy3DV>]
```

Note

It is usually best to configure the Busy DAs at a higher slot # than any pairs on any given CA. A busy specification for the A1100A SPU requires a Bus Converter address, such as 2/8.3. The defaults listed above do not include device addresses. If the address is not specified in the command line, the **CAEXR** selects the lowest available device address for each HP-IB DA.

deb[ug]

A factory test option which enables printing a trace of program execution while **CAEXR** runs. It requires source code to understand and drives the hex/LED display and, if not in "silent" mode, prints on the console.

The default is debug not enabled.

def[aults]

Specifies that the default values are to be used for the Device Adapter addresses. Default values are as shown in each category.

The default configuration is three HP-IB DAs on one CA, with pair=8.0,8.2 and busy=8.3. The default device address on each HP-IB es automatically set to the lowest available address. All other keywords default as shown in each keyword definition. On the A1100A SPU, also specify the Bus Converter address, such as 2/8.0 or 2/8.2.

errc[ount]

Instructs **CAEXR** to abandon execution and return to ISL after the specified number of errors are detected. The "nn" parameter is the number of errors in decimal. The "=" is required.

The default is to allow infinitely many errors without returning to ISL. Entering the value 0 (zero) allows an infinite number of errors.

```
errc[ount] = nn
```

erro[nly]

Causes **CAEXR** to print only error messages after the configuration is completed. It inhibits all informational messages about running status. This option does not affect the configuration dialog. Erronly is overridden by "silent" mode, which guarantees that no error or informational messages will be printed after the configuration dialog.

The default is erronly false. All informational status messages are normally printed.

ex[pert]

Reduces the interactive user interface to a terse dialog. This only affects the configuration messages. "Expert" is ignored if any other keywords are specified in the command line.

The default is expert false. The detailed user interface is the normal interface.

h[elp]

Prints information about **CAEXR** on the console before returning to ISL. It gives a brief explanation of the purpose of **CAEXR**, and enough syntax to prepare the user for the command line interface.

The default is help false, meaning that the help messages are not normally printed.

l[oop]

Specifies the number of times that **CAEXR** should repeat all configured modes, etc. One loop means that DMA transfers are performed (10 to 30 times in each 256Kb test area) * (the number of test areas in configured memory) * (once for each configured mode) * (twice, if NOSWAP is false). The "=" is required; "nn" is decimal.

The default is to loop forever. Entering a value of 0 causes **CAEXR** to loop forever.

```
l[oop]=nn
```

me[mory]

Specifies the minimum and maximum addresses (in megabytes) to be interacted with by DMA testing. DMA will be performed with all memory areas from the minimum specified megabyte to the maximum specified megabyte, inclusive. The parameters are decimal; the "=" and "," are required.

The defaults are the third megabyte through the highest megabyte of memory which is physically installed. For example, if 50 megabytes are installed, then the default range is 2,49. The counting starts at 0.

```
me[mory]=Minimum,Maximum
```

mp[x]

Configures which of three modes of operation the Channel Adapter will use when performing DMA transfers.

The three modes are subchannel, logchannel and block logchannel multiplexing.

The default is "all," meaning that all three modes are used.

```
mp[ x ]=a[ll]

    or

mp[ x ]=[b[lock][,l[ogchannel]][,s[ubchannel&]]
```

noe[rrpause]

Instructs **CAEXR** not to suspend execution when an error occurs. A hex/LED code is displayed, a message is printed on the console (if not inhibited by silent mode), and then execution continues.

The default is noerrpause false, meaning that **CAEXR** execution normally suspends for user intervention if an error occurs.

nos[wap]

Disables the automatic swapping of the master and slave roles between the Primary and Secondary DAs of a pair.

Noswap also eliminates the need for resetting the CA and the DAs between modes (if there are no errors) when subchannel mode is not used in combination with either logchannel or block logchannel modes. This means that reset will not occur during un-mixed mode operation when noswap is selected, unless there is an error. DA reset is necessary on any transition from Logchannel to Subchannel mode, or when the DA roles are swapped.

The default is noswap false, meaning that master and slave roles swap once during each loop, if enabled. At the end of the loop, the CA and DA reset occurs and the roles are effectively "unswapped," or swapped again. Therefore, a Primary DA will always have the same role at the beginning of any loop.

p[air]

Precedes the configuration data for all pairs of DAs to be configured into **CAEXR**. The "=" is required. Each BC number (<PrimaryxBC> and <SecondaryxBC>) is the decimal module # on the SMB bus; each CA number (<PrimaryxCA> and <SecondaryxCA>) is the decimal module #; each DA number (<PrimaryxDA> and <SecondaryxDA>) is the decimal slot #; each device address (<PrimaryxDV> and <SecondaryxDV>) is the HP-IB address of the device adapter.

The defaults for the three pairs are: Pair=8.0,8.2 16.0,16.2 24.0,24.2 (A1002A and 09740 SPU).

6/4.0,6/4.2 6/8.0,6/8.2(A1100A SPU).

```
p[air]=[<Primary1BC>/]<Primary1CA>.<Primary1DA>[.<Primary1DV>],
       [<Secondary1BC>/]<Secondary1CA>.<Secondary1DA>[.<Secondary1DV>]
       [<Primary2BC>/]<Primary2CA>.<Primary2DA>[.<Primary2DV>],
       [<Secondary2BC>/]<Secondary2CA>.<Secondary2DA>[.<Secondary2DV>]
       [<Primary3BC>/]<Primary3CA>.<Primary3DA>[.<Primary3DV>],
       [<Secondary3BC>/]<Secondary3CA>.<Secondary3DA>[.<Secondary3DV>]
```

Note

The defaults listed above do not include device addresses. If addresses are not specified in the command line, **CAEXR** selects the lowest available device address for each HP-IB DA.

s[ilent]

Prevents **CAEXR** from printing anything to the console after the configuration dialog. All communication from **CAEXR** to the user is with the hex/LED display.

If configuration is done through the command line, and silent mode is used, then **CAEXR** will not print anything on the console at any point in the program. No console or console path is required in this case.

The default is silent false.

Break Mode

The user can break the program at certain points by using [Control] C or [Control] Y. These user interrupts are detected after each loop completes, and for one second after any error message completes.

Note Break mode is not entered by using the console Break key.

Caution The read and write commands represent a true debug facility, which can easily destroy the state of the machine and cause a High Priority Machine Check. These debug commands should only be used by someone with a detailed knowledge of CAEXR and system internals. With explicit instructions (followed exactly), these commands are useful for examining the state of the DMA data buffers.

The following message is printed by pressing [Control] C or [Control] Y on the keyboard. It is the first indication that break mode has been entered. The following is the prompt within break mode:

```
BREAK MODE [c,d,e,h,r,s,w] (c):

Break mode commands are given below.
```

Command	Description
c	Continue - resume program execution.
d	Display - displays the current configuration values.
e	Exit - terminate the execution of CAEXR, and return to ISL.
h	Help - print a menu of break mode commands, with a brief explanation of the purpose of each.
r	(Read - reads real memory and displays the contents on the console. `Syntax: r <addr, hex>[,<number of words to read, hex>]`
s	Status - show the detailed status information for each configured device adapter.
w	Write - writes the supplied pattern to real memory at the specified address. `Syntax: w <addr, hex>,<data, hex>[,<number of words to write, hex>]`
q	Same as 'e.'
?	Same as 'h'

Diagnostic Output

The text below explains the Output Interface portion of the User Interface.

Hex/LED Display

Hex codes must appear within the range of CE80 - CEBF, using defined and undefined codes, to meet the specifications in the HP Precision Architecture RISC Chassis I/O standard. This range is not sufficient to express all of the desired information. Because of this, the hex/LED codes are followed by descriptive "parameter" values. The "class" code (CE80 - CEBF) is displayed for three seconds, followed by the stated number of "parameter" values displayed for two seconds each.

On the A1002A SPU, Hex display numbers may only be accessible through the AP.

These codes are displayed at different times during the execution of **CAEXR**. Error codes are described here as classes only, with reference to the section which contains detailed information.

Display	Source	Meaning
C580	PDC/IODC	ISL is waiting for the boot media to become ready.
CE00	ISL	**CAEXR** (or any ISL utility) is loading.
CE01	ISL	**CAEXR** (or another ISL utility) is being loaded automatically, under the control of the autoboot file.
CE13	ISL	Console input error.
CE14	ISL	Console output error.
CE15	ISL	ISL can't find the specified utility (possibly misspelled).
CE80	**CAEXR**	**CAEXR** has begun execution.
CE81 - CE8F	**CAEXR**	Reserved.
CE90	**CAEXR**	PDC call error.
CE91 - CEB8, CEBC - CEBD	**CAEXR**	**CAEXR** configuration and operation errors. See the Configuration Dialog Error message section for more detailed information.
CEB9	**CAEXR**	Data compare error. See the Data Compare Error Message section for more detailed information.
CEBA - CEBB	**CAEXR**	DMA execution errors. See the Execution Error Messages section for more detailed information.

When **CAEXR** begins exercising CAs, it displays CE80 for three seconds. Next, it displays the first loop count as *0001* for one second. Then it displays a four digit hex number for each transfer in the loop as it occurs. The least significant two digits contain a transfer number while the most significant two digits contain a loop number. When a loop concludes, **CAEXR** shows the new loop number, *0002*, displayed for one second, followed by its loop/transfer numbers. This process repeats until the specified loop count finishes (if unspecified, looping continues indefinitely). An example of hex/LED output for a loop sequence is given below:

```
0001, 0101, 0102, ... 011C, 011D, 011E, 0002, 0201, 0202, ...
```

The console output interface operates in addition to the hex/LED display output interface. The console output can provide more information than the hex/LED display.

CAEXR can run without a terminal, since **CAEXR** may disrupt terminal I/O. In addition, terminal I/O through intermittent hardware could hang the test.

Note

Since **CAEXR** ignores terminal input when in silent mode, test execution cannot be halted with [Control] [C]. The process must be allowed to run to completion.

CAEXR "silent" mode, prevents it from printing any information to the console after the configuration dialog. Configuration printing can be bypassed by entering the test parameters or the "default" parameter on the command line. The combination of command line configuration input and the "silent" mode completely inhibits console printing. In all cases, **CAEXR** outputs status information to the hex/LED display. If a DMA execution error occurs, all relevant DA and CA status information is printed.

After each transfer, the Reference, Source, Moving and Target buffers are wordwise compared. The corresponding word of each buffer is printed, for the first through fourth miscompare error. Beyond four errors, the number of words deviating from the Reference buffer are given in a quick message.

Console Error Displays

The following displays illustrate CAEXR output on the A1002A SPU.

```
*** DMA EXECUTION ERROR:  LOOP=3, Mode=Block logchannel, Transfer=29

                                Completion:      CA        Subchan   DA Stat
 Device Adapter   CA.DA.Dev MW SR SW MR    Status      Status     1st  2
 ---------------- --------- ----------   ---------   ---------   ------
 First Primary     8. 0        Y *N      00000000H  *00000000H   00H *f
 First Secondary   8. 2. 0   Y       *N   nnnnnnnnH  *nnnnnnnnH  *01H *f
 Second Primary   16. 0      Y  Y         nnnnnnnnH   nnnnnnnnH   00H  0
 Second Secondary 16. 2. 7   Y        N   nnnnnnnnH   nnnnnnnnH   01H  0
 Third Primary    24. 0      Y  Y         nnnnnnnnH   nnnnnnnnH   00H  0
 Third Secondary  24. 2. 4   Y        Y   nnnnnnnnH   nnnnnnnnH   01H  0

 First Busy        8. 6. 1   Y  -  -  -   nnnnnnnnH   nnnnnnnnH   00H  -
 Second Busy      16. 6. 5   Y  -  -  -   nnnnnnnnH   nnnnnnnnH   00H  -
 Third Busy       24. 6. 2   Y  -  -  -   nnnnnnnnH   nnnnnnnnH   00H  -

 * indicates unexpected value
```

(The line and numeric data below may optionally appear if appropriate.)

```
      Software timeout on DMA transfer  (optional display)

      Error   ceba [or cebb]   CA#  DA#  CAstat  CAstat DAstat1  DAstat2
```

The parameters which appear in the message banner, horizontal, and vertical axes are defined as follows:

```
*** BUFFER COMPARE ERROR:  Loop 5, Mode = Subchannel

 Second Moving buffer starting address = 31c26ba1H, Transfer = 13


                          Offset From Beginning of Buffers:
 Buffer Name          0009H       0013H       001cH       0025H
 -------------      ---------   ---------   ---------   ---------
 Reference        = 11551154H   11571156H   11491148H   114b114aH
 Source           = 11551154H   11571156H   11491148H   114b114aH
 Second Moving    = 11551154H   11571156H   11491148H   114b114aH
 Second Target    = c5693a96H   c5691156H   c5693a96H   114b3a96H

Number of word miscompares:  Source:  0, Moving:  0, Target:  3276

Error ceb0   pair#  ref1  ref2  MB1  MB2  offset
```

Example Session

This interface is an interactive dialog with the user through the console. It prompts the user for the BC, CA and DA address of each DA and optional devices address on each DA. If the optional device is not entered by the user, the lowest available device address is automatically selected. Directions for connecting cables are given. The user can specify the lower and upper bounds for memory to test. The default is all of available, physical memory.

The user is also allowed to specify the number of loops through the program's modes to execute before the exerciser terminates and returns to ISL. The default is to loop indefinitely.

The following screens depicts a typical interactive session which illustrates interactive user input syntax. The session begins with the appearance of the ISL banner and input prompt.

Note

To modify an entry after it has been typed at the prompt, enter Control-X and retype the entry before pressing Return.

```
ISL Revision 2726  June,1987

ISL> caexr

CAEXR Revision 2733  June,1987
CAEXR Running: CE80
```

This program provides an exhaustive check of Channel Adapter operation, including CIO bus arbitration and DMA logic.

A system-level test requires a minimum of 3 HP-IB Device Adapters for each CA tested: One PRIMARY DA - SECONDARY DA pair, and one BUSY DA. Less than this configuration will NOT fully test the CA and the CIO bus. It is not necessary to test all CAs at the same time. Note that on the HP A1100A SPU the bus converter number is required (i.e., 2/8.0). Also, if no device address is specified in response to the prompt, the lowest available device address on the HP-IB DA is automatically selected. The same thing applies if the default entry is selected.

You may enter break mode by pressing control-C or control-Y.

Press <RETURN> to continue.

Note

The normal test configuration for the 09740A and A1002A is 3 pairs available and busy on the For the A1100A, it is 4 pairs available and busy. The minimum configuration requires two pairs, but this may not saturate the bus. The maximum configuration of 12 pairs and 12 busy is for A1100A manufacturing testing. The rest of this example shows output for 09740A and A1002A.

```
Number of HP-IB DA pairs available [0-12]? Return

     Enter first PRIMARY DA's
    <CA mod #>.<DA slot #>[.<Dev Addr>] (8.0): Return

     Enter first SECONDARY DA's
    <CA mod #>.<DA slot #>[.<Dev Addr>] (8.2): Return

     Enter second PRIMARY DA's
    <CA mod #>.<DA slot #>[.<Dev Addr>] (16.0): Return

     Enter second SECONDARY DA's
    <CA mod #>.<DA slot #>[.<Dev Addr>] (16.2): Return

     Enter third PRIMARY DA's
    <CA mod #>.<DA slot #>[.<Dev Addr>] (24.0): Return

     Enter third SECONDARY DA's
    <CA mod #>.<DA slot #>[.<Dev Addr>] (24.2): Return

      .

      .

      .

Number of BUSY HP-IB DA's available [0-12]? Return

     Enter first BUSY DA's
    <CA mod #>.<DA slot #>[.<Dev Addr>] (8.3): Return

     Enter second BUSY DA's
    <CA mod #>.<DA slot #>[.<Dev Addr>] (8.0): Return

     Enter third BUSY DA's
    <CA mod #>.<DA slot #>[.<Dev Addr>] (8.0): Return

      .

      .

      .
```

CABLE CONFIGURATION:

1. Connect DA 8.0 to DA 8.2 by HP-IB.

2. Connect DA 16.0 to DA 16.2 by HP-IB.

3. Connect DA 24.0 to DA 24.2 by HP-IB.

.

.

.

Note that each connected DA pair can have no more than ONE ACTUAL HP-IB DEVICE attached to their HP-IB.

```
MEMORY CONFIGURATION:

     Enter <lower Mbyte #>,<upper Mbyte #>
          to be included in the test (2,11): Return

LOOP COUNT:

     Enter number of LOOPS through all specified modes (infinite): Return


CHANGE CAEXR PROGRAM EXECUTION MODES? [y,n] (n): Return

     Change Channel Adapter execution modes? [y,n] (n): Return

          Subchannel mode? [y,n] (y): Return

          Logchannel mode? [y,n] (y): Return

          Block mode? [y,n] (y): Return

     Pause on errors? [y,n] (y): Return

     Suppress all messages? [y,n] (n): Return

     Print error messages only? [y,n] (n): Return

     Maximum error count before return to ISL (infinite): Return

     Display software debug messages? [y,n] (n): Return
Loop 1:  Primary DA is Master

Subchannel transaction mode will be used for the following transactions...

Loop 1:  Performing 10 sets of transfers in memory area 100000H - 13fffcH
Loop 1:  Performing 10 sets of transfers in memory area 140000H - 17fffcH
Loop 1:  Performing 10 sets of transfers in memory area 180000H - 1bfffcH
Loop 1:  Performing 10 sets of transfers in memory area 1c0000H - 1ffffcH
Loop 1:  Performing 10 sets of transfers in memory area 200000H - 23fffcH
Loop 1:  Performing 10 sets of transfers in memory area 240000H - 27fffcH
Loop 1:  Performing 10 sets of transfers in memory area 280000H - 2bfffcH
Subchannel transfer:  01 02 03 04 05 06 ...
```

The following message is printed after an error occurs, to prompt the user for input:

```
Do you want to retry the transfer and continue?  [y,n] (y):
```

The program suspends until the user responds affirmatively. The *noerrpause* mode inhibits this message and the suspend condition, as shown below.

n	Disables prompting on the remaining errors associated with the current transfer, and cause the program to exit.
y	Retries the current transaction with the same data.

Note

If the test was set for an infinite loop, enter [Control] (C) or [Control] (Y) to halt the test.

Error Message Information

This listing contains all of the error messages that can be output by the Channel Exerciser program. Messages are arranged into the following three sections: Configuration Dialog messages, Execution Error messages, and Data Compare messages.

Configuration Dialog Error Messages

The following messages may be output to the hex/LED display and console when configuration dialog errors occur. The messages are listed in numerical order. The following table lists the parameter substitutions which IOMAP makes when actually sending messages to hex display or console.

Hex Display	Cons Message	Meaning
AD#	<aa>	Address number
BC#	<bb>	Bus Converter number
CA#	<cc>	Channel Adapter number
DA#	<dd>	Device Adapter number
DEV#	<ee>	Device number
MN#	<mn>	Module number
N#	<n>	Number (type specified in message)
STAT#	<stat>	Status code

HEX `ce90 STAT#`
CONSOLE `*** Error in call to Processor Dependent Code (PDC): Status = <stat>`
CAUSE A call to a PDC routine returned a bad status. It is possible that printing this line may not be successful, since output to the console requires repeated PDC calls. stat is the status returned from the PDC call.
ACTION

HEX `ce91`
CONSOLE `*** Expected '=' after kkk keyword`
CAUSE An equals sign '=' must follow immediately after the 'kkk' keyword. Keyword 'kkk' represents one of the pair, busy, errcount, loop, mpx or memory keywords on the command line interface.
ACTION

HEX `ce92`
CONSOLE `*** Expected number after kkk= keyword`
CAUSE A decimal number must follow immediately after the equals sign '=', following keyword 'kkk' on the command line interface.
ACTION

HEX `ce93`
CONSOLE `*** Unrecognized keyword kkk`
CAUSE This keyword, which was entered on the command line interface, does not match any keyword recognized by `CAEXR`.
ACTION

HEX ce94
CONSOLE *** Unrecognized MPX keyword
CAUSE The keyword entered after 'mpx=' is not 'all', or a combination of 'block,' 'logchannel' and 'subchannel.' This error occurs in the command line interface only.
ACTION

HEX ce95 <ASCII value of 'x'>
CONSOLE *** Expected 'x' as delimiter
CAUSE The command line interface expected 'x' as a delimiter, and it was not supplied.
ACTION

HEX ce96
CONSOLE *** Maximum number of DA pairs (12) exceeded
CAUSE In the command line interface, the user tried to configure more than the maximum number of DA pairs. The maximum is currently twelve pairs (twenty-four DAs).
ACTION

HEX ce97
CONSOLE *** Maximum number of Busy DAs (12) exceeded
CAUSE The user tried to configure more than the maximum allowable number of Busy DAs in the command line user interface. The maximum is currently twelve Busy DAs.
ACTION

HEX ce98
CONSOLE *** Syntax Error
CAUSE This error can occur in many places. It means that the user data entered was not in a correct or a recognizable form. Correct syntax is given in the prompt.
ACTION

HEX ce99
CONSOLE *** Number cannot be negative
CAUSE The loop count and the error count cannot be entered as negative numbers.
ACTION

HEX ce9a [BC#] CA# DA#
CONSOLE *** [BC/]CA.DA of [<bb>/]<cc>.<dd> are reserved for the console DA
CAUSE Any attempt to configure console DA will cause this message. There is currently no provision for moving the console to another path and causing this message to change.
ACTION

HEX ce9b
CONSOLE *** No Device Adapters configured
CAUSE No Device Adapters were configured in either the DA Pairs or the Busy DA section. The minimum acceptable number of DAs to configure is either one pair (two DAs) in the Pair section, or one Busy (one DA) in the Busy section.
ACTION

HEX ce9c [BC#] CA# DA#
CONSOLE *** On [BC <bb>] CA <cc>, DA <dd> has already been specified
CAUSE One of the [Bus Converter/] Channel Adapter.Device Adapter combinations which has already been configured into the exerciser was specified with the same [BC,] CA and DA numbers as the Device Adapter just entered. Every DA configured into **CAEXR** must have a unique [BC/]CA/DA address.
ACTION

HEX `ce9d [BC#] CA#`
CONSOLE `*** Invalid Mid_bus Module number [BC <bb>/] CA <cc>`
CAUSE This means that a mid_bus module number outside the range of 1 - 63 was entered. BC# is the bus converter number for the A1100A. CA# is the Channel Adapter number supplied.
ACTION

HEX `ce9e [BC#] CA#`
CONSOLE `*** Mid_bus Module [<bb>/]<cc> does not exist`
CAUSE This means that the Mid_bus module number supplied by the user CA# is valid, but is not installed in the system. BC# identifies the bus converter to which the CA is connected on the A1100A SPU.
ACTION

HEX `ce9f [BC#/] CA#`
CONSOLE `*** Mid_bus Module [bc/] <cc> is not a Channel Adapter`
CAUSE This means that the Midbus module number supplied by the user <cc> is valid, and corresponds to some module other than a Channel Adapter.
ACTION

HEX `cea0 DA#`
CONSOLE `*** Invalid Device Adapter number <dd>`
CAUSE The Device Adapter number was not between 0 and 15, inclusive.
ACTION

HEX `cea1 [BC#] CA# DA#`
CONSOLE `*** Device Adapter 'SLOW' switch (S1-7) must be down`
CAUSE The Device Adapter speed switch S1-7 must be 'down,' (toward the board surface). DIP switch S1 is at the front edge of the DA card.
ACTION

HEX `cea2 [BC#] CA# DA#`
CONSOLE `*** Device Adapter 'SCTL' switch (S1-6) must be up`
CAUSE The 'SCTL' (System Controller) switch must be set 'up' (away from the board surface). 'SCTL' is DIP switch S1-6 on the board front edge.
ACTION

HEX `cea3 [BC#] CA# DA#`
CONSOL `*** Device Adapter load resistor pack is missing`
CAUSE The resistor pack must be in place for the HP-IB to operate at the maximum speed.
ACTION

HEX `cea4 [BC#] CA# DA# DEV#`
CONSOLE `*** HP-IB device <ee> physically exists`
CAUSE HP-IB device ee is physically attached to the HP-IB of the Device Adapter being configured. The user must specify a different HP-IB device.
ACTION

HEX `cea5 [BC#] CA# DA#`
CONSOLE `*** Too many HP-IB devices on DA pair/busy`
CAUSE A maximum of one extra HP-IB device is permissible on the HP-IB connecting a Device Adapter pair. **CAEXR** has detected more than one device attached. An HP-IB listed as busy must have at least one device address free of any devices. **CAEXR** has determined that no device address is available.
ACTION

HEX `cea6 <v1> <v2>`
CONSOLE `*** Minimum test value v1 exceeds minimum test value v2`
CAUSE The first integer entered must be less than or equal to the second integer entered.
ACTION

HEX	**cea7 1 <max PM> <min CM> <max CM>**
CONSOLE	***** Testable memory is 1 through <max PM> (in Mbytes)**
CAUSE	At least one of the memory configuration parameters entered by the user (<min CM> or <max CM>) is outside the range of the first megabyte through the highest megabyte physically installed (<max PM>), inclusive.
ACTION	

HEX	**cea8**
CONSOLE	***** No modes configured**
CAUSE	At least one mode (Subchannel, Logchannel or Block) must be configured for **CAEXR** to perform any meaningful testing.
ACTION	

HEX	**cea9**
CONSOLE	***** Number out of range**
CAUSE	The number entered is too large or too small to be accepted. A correct value must be within the range shown in the prompt itself.
ACTION	

HEX	**ceaa [BC#] CA# DA#**
CONSOLE	***** On [BC <bb>,] CA <cc>, Device Adapter <dd> failed selftest**
CAUSE	The DA Sense Register PST bit is not set, indicating that the DA selftest failed last time it was run.
ACTION	

HEX	**ceab [<BC#>] CA# DA#**
CONSOLE	***** On [BC<bb>,] CA <cc>, Device Adapter <dd> is not 'Ready For Command'**
CAUSE	The Sense Register RFC bit is not set, meaning that the DA is not able to accept another command. It should be ready at this time.
ACTION	

HEX `ceac [<BC#>] CA# DA#`
CONSOLE `*** On [BC<bb>,] CA <cc>, Device Adapter <dd> DMA timeout`
CAUSE A DMA transfer initiated for this DA failed to complete within a reasonable period of time (currently one second). The third word of the final entry in the CA link status list does not point to the last quad in the chain. This message applies during the configuration dialog, and to setup before and between transfers.
ACTION

HEX `cead BC#`
CONSOLE `*** Invalid Bus Converter number <bb>`
CAUSE A Bus Converter module number <bb> outside the range 1-63 was entered.
ACTION

HEX `ceae BC#`
CONSOLE `*** Bus Converter Module <bb> does not exist`
CAUSE The Bus Converter Module number <bb> is valid but the corresponding module is not installed in the system.
ACTION

HEX `ceaf [BC#] CA# DA# (dev addr)`
CONSOLE `*** On [BC <bb>,] CA <cc>, Device Adapter <dd>, invalid device address <aa> specified`
CAUSE The specified device address is outsidethe valid range for device addresses.
ACTION

HEX `ceb0 <BC#> CA# DA# STAT#`
CONSOLE `*** On [BC <bb>,] CA <cc>, Device Adapter <dd> status error: Status = <stat>`
CAUSE The Device Adapter status does not match the expected value, after a DMA transfer was performed. This message is generated during the configuration dialog and setup sections.
ACTION

HEX `ceb1 [<BC#>] MN# STAT#`
CONSOLE `*** [BC <bb>,] Module Number <dd> module error: Status = <stat>`
CAUSE The module error bit is set in the Channel Adapter status register or an error was detected in the memory controller. This message is currently generated either during the configuration dialog section or immediately after a DMA execution error or memory compare error.
ACTION

HEX `ceb2 [<BC#>] CA# STAT#`
CONSOLE `*** [BC<bb>,] Channel Adapter <cc> timeout: Status = <stat>`
CAUSE The CA io_stat register ca_ready bit did not become set (ready) within a reasonable interval (currently one second). This prevented a DMA transfer from being started by software.
ACTION

HEX `ceb3 [<BC#>] CA# DA# STAT#`
CONSOLE `*** On [BC <bb>,] CA <cc>, subchannel <dd> not ready: Status = <stat>`
CAUSE On the CA, this subchannel's status register subc_ready bit did not come ready within a reasonable period of time (currently one second). This prevented a DMA transfer from being started by software.
ACTION

HEX `ceb4 [<BC#>] CA# DA#`
CONSOLE `*** On [BC <bb>,] CA <cc>, Device Adapter <dd> does not exist`
CAUSE The specified Device Adapter number (DA#) does not correspond with a DA installed on the specified CA and BC (A1100A only). The Device Adapter that the user wishes to configure might be installed on a different BC or CA than was specified.
ACTION

HEX `ceb5 [<BC#>] CA# DA#`
CONSOLE `*** On [BC<bb>] CA <cc>, Device Adapter <dd> is not an HP-IB DA`
CAUSE The Device Adapter number entered corresponds to some type of DA which is not a correct HP-IB DA. For instance, it might be a MUX or LANIC card.
ACTION

HEX `ceb6`
CONSOLE `*** Fatal error--internal buffer(s) overwritten`
CAUSE **CAEXR** has suffered an internal data management problem. If it is not a bad copy of **CAEXR**, then it must be a software malfunction, or a seriously out-of-control DMA transfer. Gather data for a "service request" report.
ACTION

HEX `ceb7 [<BC#>] CA# DA#`
CONSOLE `*** On [BC<bb>,] CA <cc>, Busy Device Adapter <dd> DMA timeout`
CAUSE The Busy DA failed to complete its DMA transfer before the software timer expired.
ACTION

HEX `ceb8 AD# STAT# AD# STAT#`
CONSOLE `*** Secondary address/status error: Expected <aa>/<stat>, read <aa>/<stat>`
CAUSE In Block Logchannel mode (only), the slave reads a secondary address (HP-IB secondary command) between each block of data read or written. Both the secondary address itself, and the status of the read obtaining the secondary address, are checked for correct values. This message is printed if either (or both) of the values are wrong.
ACTION

HEX ceb9 ...
CONSOLE *** BUFFER COMPARE ERROR: ...
CAUSE See the Data Compare Error Message section for the meaning of this error message.
ACTION

HEX ceba ...
CONSOLE *** DMA EXECUTION ERROR: ...
CAUSE See the Execution Error Message section for the meaning of this error message.
ACTION

HEX cebb ...
CONSOLE *** DMA EXECUTION ERROR: ...
CAUSE See the Execution Error Message section for the meaning of this error message.
ACTION

HEX cebc [<BC#>,] CA# DA# STAT#
CONSOLE *** On [BC <bb>,] CA <cc>, Subchannel <dd> ARQ timeout: Status = <stat>
CAUSE The one second timeout timer expired before ARQ came true in the DA sense register.
ACTION

HEX cebd [BC#] CA# DA# STAT#
CONSOLE *** On [BC, <bb>] CA <cc>, Subchannel <dd> was not destroyed: Status = <stat>
CAUSE After ARQ was seen in the DA sense register, the attempt to Destroy SubChannel on the DA by Direct I/O failed (meaning that no Sub Channel Destroyed message was received back from the DA).
ACTION

HEX	`cebe N#`
CONSOLE	`*** Maximum error count <n> exceeded, program will abort`
CAUSE	Maximum number of errors specified in Errcount parameter was exceeded.
ACTION	

Execution Error Messages

The following describes how execution errors are shown on the Hex and Console displays respectively. Message format is defined in the syntax example below.

CEBA	A normal execution error, as a result of a CA or DA status being incorrect at the end of a transfer.
CEBB	Similar to CEBA, except that it was triggered by the 3 second timer on the software wait loop. The DMA transfers ordinarily complete in approximately 100-200mS.

Syntax example: (displayed for the first error only)

```
CEBx   [BC#]  CA#   DA#  CAstat  CAstat  DAstat1  DAstat2
```

After starting the DMA transfers for all of the configured Device Adapters, software waits until all transfers are complete. Software then checks for a successful completion status for each DA pair. In addition, the quad program status is checked for each configured Busy DA. If a bad status is found for any configured DA or its corresponding CA, then an execution error message is printed. The following example shows a typical error message on a 9740A and A1002A SPU. On the A1100A SPU, each entry in the "CA.DA.Dev" column also contains the appropriate Bus Converter number and slant (/).

```
*** DMA EXECUTION ERROR:  LOOP=3, Mode=Block logchannel, Transfer=29

                                Completion:      CA         Subchan    DA Stat
 Device Adapter    CA.DA.Dev  MW SR SW MR     Status       Status     1st  2
-----------------  ---------  ----------    ---------    ---------   ------
First Primary       8. 0          Y *N       00000000H   *00000000H   00H *f
First Secondary     8. 2. 0   Y       *N     nnnnnnnnH   *nnnnnnnnH   *01H *f
Second Primary     16. 0      Y  Y          nnnnnnnnH    nnnnnnnnH    00H  0
Second Secondary   16. 2. 7   Y             nnnnnnnnH    nnnnnnnnH    01H  0
Third Primary      24. 0      Y  Y          nnnnnnnnH    nnnnnnnnH    00H  0
Third Secondary    24. 2. 4   Y             nnnnnnnnH    nnnnnnnnH    01H  0

First Busy          8. 6. 1   Y  -  -  -    nnnnnnnnH    nnnnnnnnH    00H  -
Second Busy        16. 6. 5   Y  -  -  -    nnnnnnnnH    nnnnnnnnH    00H  -
Third Busy         24. 6. 2   Y  -  -  -    nnnnnnnnH    nnnnnnnnH    00H  -

  * indicates unexpected value
```

(The line and numeric data below may optionally appear if appropriate.)

```
Software timeout on DMA transfer (optional display)

Error   ceba [or cebb]   CA#  DA#  CAstat  CAstat DAstat1  DAstat2
```

The parameters which appear in the message banner, horizontal, and vertical axes are defined as follows:

LOOP	The decimal number of the loop in which this error occurred.
Transfer	This is the decimal number of the DA pair transfer in the memory area under test which completed or failed before the error was detected.
Mode	The type of DMA transfers which are being done at this time. Possible choices are: Subchannel, Logchannel and Block Logchannel.
[BC/]CA.DA.Dev	The physical [Bus Converter module] Channel Adapter module, Device Adapter slot numbers and HP-IB address for this DA. The [BC,] CA and DA numbers and HP-IB address are the values which the user entered or defaulted in the configuration dialog.
Completion	This parameter indicates whether the four phases of the DMA transactions for this transfer completed or not. A DMA can be complete and still have other errors. Each phase of the transfer is listed below. Only the MW phase applies to the Busy DA.
MW	The Master Write phase of the first transaction of the transfer. For the DA of the pair which is currently Master (Controller In Charge of the HP-IB), this shows whether the DMA write from main memory (source buffer) to the Slave DA was completed or not. In the example, a 'Y' in the MW column for the first Secondary DA shows that this transfer was completed. This also shows that the Secondary DA was Master at the time of the error.
SR	The Slave Read phase of the first transaction of the transfer. For the DA of the pair which is currently Slave (not Controller In Charge), this shows whether the DMA read from the Master DA to main memory (to the area under test) was completed or not.
SW	The Slave Write phase of the second DMA transaction of the transfer. This shows whether the Slave DA DMA write from main memory (from the area under test) to the Master DA was completed or not.

MR	The Master Read phase of the second DMA transaction of the transfer. It shows whether the Master DA DMA read from the Slave DA to main memory (to the target buffer) was completed or not.
CA Status	The value of the Channel Adapter status register, read after this transfer completed.
Subchan Status	The final Subchannel Status, extracted from the link status list for the final DMA transfer for this DA. If all of the phases of the DMA transfer were not completed for this DA, this word is 0.
DA Status	The final Device Adapter status for the final phase of this DMA transfer. If all of the phases of the DMA transfer were not completed for this DA, this word is ffH. In Block Logchannel mode, the Master Write transaction does exist. It comes up as 'ffH', without a star, but does not generate an error.
Software timeout on DMA transfer	This message is only printed if the 3 second software timer expired before the DMA transfer completed. Accompanies the cebb execution error status.
cebx	*ceba* is the error code for an ordinary execution error. *cebb* denotes that an execution error occurred, along with a pair DMA transfer failing to complete in within three seconds.

Data Compare Error Messages

The following explains both hex/LED and console display Data Compare Error Messages respectively. An example of the hex/LED display output appears below followed by an explanation of how these messages are handled through the console.

```
ceb0  Pair#  Ref1  Ref2  MBadr1  MBadr2  Offset
```

Code	Meaning
ceb0	Error code for a Data Compare Error.
Pair#	Pair number. Whether the first, second or third pair buffers had the error.
Ref	The contents of the Reference Buffer at the error offset, high- and low-order 16 bits.
Mbadr	Moving Buffer address: The high and low order 16 bits of the starting address of the Moving Buffer in main memory.
Offset	The offset into the buffers where the first miscompare error was detected.

The Source, Moving and Target data buffers used in the DMA transfers for each DA pair are compared to the Reference buffer to test for incorrect data. This testing is done regardless of whether or not there is an execution error.

There is only one Reference buffer and one Source buffer. There is one Moving buffer and one Target buffer for each configured DA pair.

For each configured DA pair, each word of the Source, Moving and Target buffers are compared against the corresponding word of the Reference buffer. The first four words which do not compare cause the corresponding words from all four buffers to be printed for user examination, as shown in the example below. If miscompares occur for more than four words, then the number of words in error are noted in a message beneath the display.

Each word of each Target buffer is initialized to C5693A96 hex before each DMA transfer starts. Each word of each Moving Buffer is initialized to B5AE2D8A hex before each DMA transfer starts.

```
*** BUFFER COMPARE ERROR:  Loop 5, Mode = Subchannel

  Second Moving buffer starting address = 31c26ba1H, Transfer = 13

                          Offset From Beginning of Buffers:
   Buffer Name          0009H       0013H       001cH       0025H

   Reference        = 11551154H   11571156H   11491148H   114b114aH
   Source           = 11551154H   11571156H   11491148H   114b114aH
   Second Moving    = 11551154H   11571156H   11491148H   114b114aH
   Second Target    = c5693a96H   c5691156H   c5693a96H   114b3a96H

Number of word miscompares:  Source:  0, Moving:  0, Target:  3276

ceb0   pair#  ref1  ref2  MB1  MB2  offset
```

Moving Buffer

The 32Kb location in the memory area under test which is the intermediate stop for the test data. Data is transferred by DMA through the DA pair from the Source buffer to the Moving buffer. The data is then moved by DMA back through the DA pair, from the Moving buffer to the Target buffer. All of the buffers are then wordwise compared. This buffer is initialized to b5ae2d8aH before the DMA transfers begin.

Starting address

The absolute address of the beginning of the Moving buffer. This address is randomly selected for each transfer to be some value within the memory area under test.

Offset

The wordwise displacement from the start of each of the buffers to a word containing an error. Visual inspection can then show which buffer(s) contain an error, and what type of data corruption was experienced.

Source buffer

A reserved 32Kb area in low memory, which contains the semi- random data copied from the Reference buffer. This data should be unique for each transfer in CAEXR. This data is read by DMA from main memory by the DA in the pair which is performing the master role, and written out the HP-IB to the DA performing the slave function. This slave DA reads the data from the HP-IB, and writes it with DMA to the Moving buffer in the memory area under test. Only one Source buffer exists for the use of all DA pairs.

Target buffer

A reserved 32Kb area in low memory, which is the ultimate destination of the data being moved around by the DA pair. Before the DMA transfers occur, each word of this buffer is initialized to c5693a96H.

After the Moving buffer is written to by the DA pair, the slave role DA of the pair reads the Moving buffer by DMA, and write the data to the HP-IB. The master role DA reads the HP-IB, and writes this data to the Target buffer by DMA.

Reference buffer	A reserved 32Kb area in low memory, which is initialized to semi-random data before each DMA transfer sequence begins. This data is copied to the Source buffer before the DMAs occur. The DAs do not even read from the Reference buffer by DMA, so this data is the most accurate copy of the original data available.
Number of word miscompares	The total number of words in each the Source, Moving and Target buffers which do not compare with the corresponding words in the Reference buffer. This includes all errors, whether or not they are listed in the table of errors.

Contents

8

Bus Converter Diagnostic (BCDIAG)

Introduction

BCDIAG is a diagnostic program designed to test the A1126A Bus Converter. It does not test any other standalone or system internal Bus Converter. BCDIAG is launched from the ISL (only), which means the operating system is down. The system is completely unavailable to the customer while BCDIAG is running. BCDIAG can be loaded from the ISL boot directory or from the support tape.

When BCDIAG is run in the default mode it displays all BCs connected to the system, their paths, hardware and software model numbers, type IDs, and revision numbers where applicable. It also selftests the BCs, tests the forward progress screens, and tests the power fail interrupt mechanism.

Error messages are printed for any component that fails a test. The identify and the tests may be limited to an individually specified path, or select only a certain set of tests. BCDIAG contains five test modes: Identify, Selftest, Interrupt Test, FPS Test, and DMA Test.

Defects and Enhancements

Submit defect reports and enhancement requests concerning this diagnostic through the STARS database referencing product number 30345-10003.

Minimum Configuration

The minimum hardware configuration required to troubleshoot the A1126A Bus Converter consists of the following:

- Bus Converter (A1126A)
- Functional computer system
- Verified power source
- Support tape

Functional Overview

When invoked with the default option BCDIAG will: display all Bus Converters plugged into the system, their paths, their hardware and software model numbers, their type IDs and revision numbers where applicable. The configuration will be determined by PDC calls. It will also selftest the Bus Converters , test the forward progress screens and test the power fail interrupt mechanism. Error messages will be printed for any component which fails a test. The user may also limit the identify and the tests to an individually specified path or choose only a certain set of tests.

The diagnostic takes about 30-45 seconds to run. Initially, the selftest in IODC is accessed and executed. After selftest has completed, there is a delay of approximately 15 seconds, while the bus converters are being initialized. Next, all selected sections of the diagnostic are executed.

Help Facility

BCDIAG has an online help facility, which is invoked by typing the following:

```
ISL>bcdiag help
```

The following will then be echoed to the screen:

```
                                BCDIAG "HELP" FACILITY
The BCDIAG utility has a command line interface and an interactive
user interface:

BCDIAG--Command Line Syntax:
   deb[ug] - output debug messages and LED codes
   def[aults] - use defaults (instead of interactive prompting)
   errc[ount]=<nn> - exit after specified number of errors occur
   erro[nly] - display only error messages
   h[elp] - display this help message
   l[oop]=<nn> - specifies number of times through main program loop
   n[oerrpause] - disable pausing on errors
   p[ath]= <n[/n.../n].n ... .n> - specific path to test (instead of all paths)
   s[ilent] - disable all printing
   t[ests]=[a[ll] [,s[elftest]] [,interr[upt]] [,id[entify]] [,f[ps]] [,d[ma]]
    - test type
   intera[ctive_test] - run the selftest interactively: to be used ONLY in
    conjunction with tests=selftest option.

BCDIAG--Interactive Interface:
   To enter the interactive interface enter "bcdiag"

Press <RETURN> to exit.

ISL>
```

Operating Instructions

BCDIAGcan be run via one of two user input interfaces:

1. Command line
2. Interactive

Command Line User Input Interface

The command line is the information which is typed at the ISL prompt before hitting carriage return. To run BCDIAG, the string bcdiag is typed followed by optional parameters which the user wishes to pass into the program. The keywords and syntax of these parameters is given in the following section.

One purpose of the command line input is to bypass all or part of the interactive dialog. If any parameter is entered on the command line, BCDIAG will use all the supplied values, with the balance of the values being set to their defaults.

If any keyword is specified on the command line, the interactive user interface is bypassed. If no keyword is specified, the interactive user interface will be invoked.

Using the command line input to bypass the interactive dialog does not affect the printing of status or error messages on the console. To limit console output to error messages only (status and error messages are still presented on the hex/LED display), the **erronly** mode must be specified on the command line. To limit message output to the hex/LED display only, **silent** mode can be specified.

The autoboot file on any boot medium can be changed to run BCDIAG automatically, when the system boots from that device. All parameters can be specified using the command line input feature. **Silent** mode can be specified to keep messages from being printed to the console. This means that BCDIAG can be run without working or present console path hardware.

Command Line Input Syntax

Each command line keyword is listed and described below. The keywords can be entered in abbreviated form or full word, the optional part of the keyword is shown inside square brackets [ ] with the required part outside the brackets.

Several keywords are followed immediately by a required equal sign =. Optional and required parameters are as shown for each keyword. All numeric input is in decimal. All periods and commas are required unless they are contained within square brackets [ ].

Keywords:

def[ault]
errc[ount]
erro[nly]
h[elp]
l[oop]
n[errpause]
p[ath]
intera[ctive_test]
s[ilent]
t[est]

def[ault]	Default values are used, interactive interface is bypassed.
errc[ount]=nn	BCDIAG halts and returns to ISL after nn errors are detected. Default or nn value of 0 allows infinite number of errors.
erro[nly]	BCDIAG prints only error messages and inhibits informational messages. Erronly is overridden by silent mode. Default is erronly false, all messages are printed.
h[elp]	Puts information about BCDIAG on the console.
l[oop]=nn	Specifies the number of times the selected tests are repeated. Default is to run tests once, an nn value of 0 allows an infinite number of loops.
n[oerrpause]	Instructs BCDIAG not to suspend execution when an error occurs. A hex code is displayed and a message is sent to the console and execution continues. Default is false, execution suspends for user intervention.
p[ath]=n (9000/840 and 9000/825/835) p[ath]=n/n (9000/850/855)	Restricts BCDIAG operation to the specified (n or n/n) I/O path. Default is for BCDIAG to run on all A1126A Bus Converters connected to the system.
intera[ctive_test]	This keyword can ONLY be used with SELFTEST. It allows the user to specify values to pattern test the registers. Default is false.
s[ilent]	Prevents BCDIAG from putting anything on the console after the interactive dialog. Hex display is still used. Default is false.
t[ests]=[id[entify]][,s[elftest] t[ests]=[,f[ps]][,interr[upt]][,d[ma]] t[ests]=a[ll]	Specifies which tests are to be run. All selects the four available tests. Default is to run the IDENTIFY test only.

Interactive User Input Interface

This interface is an interactive dialog with the user through the console. It prompts the user for the same information that could be entered from the command line.

Interactive User Input Syntax

```
ISL> bcdiag

BCDIAG: Revision A.01.06; August 8, 1989

Bcdiag has the ability to identify and test A1126A Bus
Converters. It can execute the ROM based selftest, test
the Forward Progress Screens and interrupts.

Without changing any parameters, this diagnostic will identify every
attached A1126A Bus Converter, running selftest and a default set of tests
on them.

You may enter break mode by pressing control-C or control-Y.

You may press control-X to erase what you just typed in.


STATUS: The diagnostic will now scan the system and configure
all modules EXCEPT A1126A Bus Converters.
This could take 15 seconds.

STATUS: The diagnostic will now configure A1126A bus converters,
and run selftest on them.

 Testing BC at path = 2

 Testing BC at path = 2/24

General/Cursory BC1 Testing
   Step 1 - MB port General Tests
   Step 2 - NIO port General Tests
        MB port PASSED General Test
        NIO port PASSED General Test

Testing IODC_DATA/IODC_ADDRESS Register
   Step 1 - MB port IODC_DATA/IODC_ADDRESS Register Tests
   Step 2 - NIO port IODC_DATA/IODC_ADDRESS Register Tests
        MB port IODC_DATA Register tested OK
        NIO port IODC_DATA Register tested OK
```

```
Testing IO_COMMAND Register
   Step 1 - MB port IO_COMMAND Register Tests
   Step 2 - NIO port IO_COMMAND Register Tests
        MB port IO_COMMAND Register tested OK
        NIO port IO_COMMAND Register tested OK

Testing IO_STATUS Register
   Step 1 - MB port IO_STATUS Register Tests
   Step 2 - NIO port IO_STATUS Register Tests
        MB port IO_STATUS Register tested OK
        NIO port IO_STATUS Register tested OK

Testing IO_CONTROL Register
   Step 1 - MB port IO_CONT Reg Tests {Non Destructive}
   Step 2 - MB port IO_CONT Reg Tests {Destructive}
   Step 3 - NIO port IO_CONT Reg Tests {Non Destructive}
   Step 4 - NIO port IO_CONT Reg Tests {Destructive}
   Step 5 - BC1 Mode Functionality Tests
        MB port IO_CONT PASSED Non Destructive Tests
        MB port IO_CONTROL PASSED Destructive Tests
        NIO port IO_CONT PASSED Non Destructive Tests
        NIO port IO_CONTROL PASSED Destructive Tests
        BC1 MODE Functionality tested OK

Testing IO_ERR_SADD/IO_ERR_MADD Registers
   Step 1 - Prompting ERR_SADD/MADD patterns {interactive}
   Step 2 - MB port ERR_SADD/MADD Reg Tests {Unfrozen}
   Step 3 - MB port ERR_SADD/MADD Reg Tests {Frozen}
   Step 4 - NIO port ERR_SADD/MADD Reg Tests {Unfrozen}
   Step 5 - NIO port ERR_SADD/MADD Reg Tests {Frozen}
        MB port ERR_SADD/MADD PASSED Unfrozen Tests
        MB port ERR_SADD/MADD PASSED Frozen Tests
        NIO port ERR_SADD/MADD PASSED Unfrozen Tests
        NIO port ERR_SADD/MADD PASSED Frozen Tests

Testing IO_IO_LOW/IO_IO_HIGH Registers
   Step 1 - Prompting IO_LOW/HIGH patterns {interactivv}
   Step 2 - MB port IO_IO_LOW/IO_IO_HIGH Registers Tests
   Step 3 - NIO port IO_IO_LOW/IO_IO_HIGH Registers Tests
        MB port IO_LOW/HIGH Registers tested OK
        NIO port IO_LOW/HIGH Registers tested OK

Testing PENDING read (0 - 3) Registers
   Step 1 - Prompting PENDING read patterns {interactive}
   Step 2 - NIO port PENDING read (0 - 3) Registers Tests
        PENDING read (0 - 3) Register tested OK
```

```
Testing BUSY_WRITE/BUSY_READ Registers
   Step 1 - Prompting BSY_WR/BSY_RD patterns {interactive}
   Step 2 - MIO port BUSY_WRITE/BUSY_READ Registers Tests
        BUSY_WRITE/BUSY_READ Register tested OK

BC1 LBQ/BBQ Shuffle Testing
   Step 1 - Prompting LBQ/BBQ patterns {interactiv}
   Step 2 - MB port LBQ/BBQ Shuffle Tests
        MB port PASSED LBQ/BBQ Shuffle Test

Do you wish to modify any program parameters? [y,n] (n):
n
```

Note

If you had answered y to this prompt, instead of n, you would have been prompted in a manner similar to the following:

```
Change tests to be executed?  [y,n] (n):  y

    Identify?  [y,n] (y):  y

    Selftest?  [y,n] (y):  y

        Execute Selftest Immediately?  [y,n]  (n):  n

    Interrupt?  [y,n] (y):  y

    Fps? [y,n] (y):  y

    DMA? [y,n] (y):  y

Test single path only? [y,n] (n):  n

Enter number of LOOPS [<n>,0=infinite]  (1):  1

Change miscellaneous program parameters? [y,n] (n):  y

    Pause on errors? [y,n] (y):  y

    Maximum error count before returning to ISL? (infinite):  1

    Suppress all messages?  [y,n] (n):  n

    Print error messages only? [y,n] (n):  n

    Display software debug messages?  [y,n]  (n):  n
```

```
Identify:  Loop 1:

                                                        Type    SW    HW   Revisions
Path         Component Name                              ID    Mod   Mod   Hdwr   Firm
------------ ------------------------------------------ ----  ----  ----  ----   ----
2            Bus Converter: NOT an A1126A
2/24         A1126A Bus Converter MIDBUS PORT            7H    CH     5     0
             A1126A Bus Converter HP-PB PORT             7H    CH   256     0


Selftest:  Loop 1:

General/Cursory BC1 Testing
   Step 1 - MB port General Tests
   Step 2 - NIO port General Tests
        MB port PASSED General Test
        NIO port PASSED General Test

Testing IODC_DATA/IODC_ADDRESS Register
   Step 1 - MB port IODC_DATA/IODC_ADDRESS Register Tests
   Step 2 - NIO port IODC_DATA/IODC_ADDRESS Register Tests
        MB port IODC_DATA Register tested OK
        NIO port IODC_DATA Register tested OK

Testing IO_COMMAND Register
   Step 1 - MB port IO_COMMAND Register Tests
   Step 2 - NIO port IO_COMMAND Register Tests
        MB port IO_COMMAND Register tested OK
        NIO port IO_COMMAND Register tested OK

Testing IO_STATUS Register
   Step 1 - MB port IO_STATUS Register Tests
   Step 2 - NIO port IO_STATUS Register Tests
        MB port IO_STATUS Register tested OK
        NIO port IO_STATUS Register tested OK

Testing IO_CONTROL Register
   Step 1 - MB port IO_CONT Reg Tests {Non Destructive}
   Step 2 - MB port IO_CONT Reg Tests {Destructive}
   Step 3 - NIO port IO_CONT Reg Tests {Non Destructive}
   Step 4 - NIO port IO_CONT Reg Tests {Destructive}
   Step 5 - BC1 Mode Functionality Tests
        MB port IO_CONT PASSED Non Destructive Tests
        MB port IO_CONTROL PASSED Destructive Tests
        NIO port IO_CONT PASSED Non Destructive Tests
        NIO port IO_CONTROL PASSED Destructive Tests
        BC1 MODE Functionality tested OK
```

```
Testing IO_ERR_SADD/IO_ERR_MADD Registers
   Step 1 - Prompting ERR_SADD/MADD patterns {interactive}
   Step 2 - MB port ERR_SADD/MADD Reg Tests {Unfrozen}
   Step 3 - MB port ERR_SADD/MADD Reg Tests {Frozen}
   Step 4 - NIO port ERR_SADD/MADD Reg Tests {Unfrozen}
   Step 5 - NIO port ERR_SADD/MADD Reg Tests {Frozen}
        MB port ERR_SADD/MADD PASSED Unfrozen Tests
        MB port ERR_SADD/MADD PASSED Frozen Tests
        NIO port ERR_SADD/MADD PASSED Unfrozen Tests
        NIO port ERR_SADD/MADD PASSED Frozen Tests

Testing IO_IO_LOW/IO_IO_HIGH Registers
   Step 1 - Prompting IO_LOW/HIGH patterns {interactivv}
   Step 2 - MB port IO_IO_LOW/IO_IO_HIGH Registers Tests
   Step 3 - NIO port IO_IO_LOW/IO_IO_HIGH Registers Tests
        MB port IO_LOW/HIGH Registers tested OK
        NIO port IO_LOW/HIGH Registers tested OK

Testing PENDING read (0 - 3) Registers
   Step 1 - Prompting PENDING read patterns {interactive}
   Step 2 - NIO port PENDING read (0 - 3) Registers Tests
        PENDING read (0 - 3) Register tested OK

Testing BUSY_WRITE/BUSY_READ Registers
   Step 1 - Prompting BSY_WR/BSY_RD patterns {interactive}
   Step 2 - NIO port BUSY_WRITE/BUSY_READ Registers Tests
        BUSY_WRITE/BUSY_READ Register tested OK

BC1 LBQ/BBQ Shuffle Testing
   Step 1 - Prompting LBQ/BBQ patterns {interactiv}
   Step 2 - MB port LBQ/BBQ Shuffle Tests
        MB port PASSED LBQ/BBQ Shuffle Test
```

```
Interrupt:   Loop 1:


      **********************************
Please turn off the power switch on the A1126A Box
      **********************************

And then hit RETURN:
      **********************************
Please turn on the power switch on the A1126A Box
      **********************************

And then hit RETURN:


      **********************************
Please turn off the power switch on the A1126A Box
      **********************************

And then hit RETURN:
      **********************************
Please turn on the power switch on the A1126A Box
      **********************************


And then hit RETURN:


Fps:   Loop 1:

Test Busy Write/Read Master register on NIO port
Test all Pending Registers on NIO port
Test no Slave Ack on NIO Port
Test intermittent errors on NIO Port
Test Io_Err_Madd/Sadd on Midbus Port
```

```
DMA:   Loop 1:

Performing DMA testing with the following HPPB cards:
    HPPB card in A1126A slot 4

Turning on DMA for HPPB cards
Performing processor writes to the lower port iodc register
Performing processor reads to the lower port iodc register
Performing processor writes and reads of the lower port iodc register
Checking that HPPB cards have read and written the correct data
Inverting the HPPB card read data, and checking that it is written correctly
Again, inverting the HPPB card read data and checking writes
Stopping HPPB cards by putting an EOC in the link quad
Turning off BC Remote Port
Checking to make sure that BC and HPPB cards did not log any errors
Turning on DMA for HPPB cards while remote port is off
Checking to see that the cards log 'no slave acknowledge' as they should
Putting BC Remote Port in exclude mode
Turning on DMA for HPPB cards

Loop 1 (1H) complete.


Do you want to exit this program and return to ISL? [y,n] (n): y

BCDIAG Exiting Successfully.

ISL>
```

Detailed Test Descriptions

BCDIAG can be run in five different test modes:

- Identify
- Selftest
- Interrupt Test
- FPS Test
- DMA Test

Identify

The identify test attempts to identify each Bus Converter in every I/O path (or specified path) if the Bus Converter is identifiable. The information printed includes:

1. I/O path to the component
2. Component name
3. Component ID number
4. Component software model number (if applicable)
5. Component hardware model number (if applicable)
6. Firmware revision
7. Hardware revision

Selftest

The selftest is implemented as an IODC entry test. BCDIAG calls the entry test, passes input from the user to IODC and displays the status and error messages (on the system console) passed to it by IODC. When selftest is used in conjunction with the `interactive_test` option, the user can specify the number of times to execute each step in the selftest, and the patterns to be used to test the registers. The selftest:

1. Tests accessibility of the registers on both Mid-Bus and HP-PB cards.
2. Pattern tests the registers
3. Tests loopback on the Mid-Bus card using the shuffle command. It should be able to isolate the FRUs

Interrupt Test

The interrupt test checks the compatibility of the Bus Converter to synthesize interrupts, detect and log errors. This is the only test that requires user intervention during the test. The user needs to turn the power switch ON and OFF as prompted by the program.

FPS Test

The FPS tests the Forward Progress Screen algorithm. It tests the pending registers, busy read/write registers, I/O error MADD/SADD registers on the Mid-Bus port, and the capability of the HP-PB port. It exercises the BC and detects intermittent faults by generating DMA transactions from HP-PB modules to memory. This test requires a MAP/PSI card to be installed.

DMA Test

The DMA test performs processor reads and/or writes of the IODC register, and checks the remote port for the Bus Converter, to ensure that no errors were logged and to see that the cards under test logged no `slave acknowledge`.

Error Messages

Error messages are displayed as informational messages on the console or Hex code messages on the BC Hex display. The Hex display displays parameter values along with the error message by; displaying the class code (CE80 to CEBF) for three seconds, followed by the stated number of parameter values displayed for two seconds each.

Console Display and Hex Error Codes:

HEX	`ce90 stat`
CONSOLE	`***Error in call to the Processor Dependent Code (PDC): Status = stat`
CAUSE	A call to a PDC routine returned a bad status.
HEX	`ce91`
CONSOLE	`***Expected '=' after "kkk" keyword`
CAUSE	An equal sign = must follow immediately after kkk (keyword).
HEX	`ce92`
CONSOLE	`***Expected number after "kkk="`
CAUSE	A decimal number must follow kkk= on the command line.
HEX	`ce93`
CONSOLE	`***Unrecognized keyword "kkk"`
CAUSE	Keyword kkk is not valid.
HEX	`ce94`
CONSOLE	`***Unrecognized tests keyword kkk`
CAUSE	Keyword kkk after *test=* is not *all* or there is a comma missing between test names.

HEX	`ce95`
CONSOLE	`***Expected 'c' as delimiter`
CAUSE	Command line expected 'c' but another character was supplied.

HEX	`ce98`
CONSOLE	`***Syntax error`
CAUSE	Data entered not in correct or recognizable form.

HEX	`ce99`
CONSOLE	`***Number cannot be negative`
CAUSE	Loop count and error count cannot be entered as negative numbers.

HEX	`ce9d n`
CONSOLE	`***Invalid module number n`
CAUSE	Module number outside the range 1 to 63 was used. *n* is the module number supplied by the user.

HEX	`ce9e [bc]n`
CONSOLE	`***Module [bc/]n does not exist`
CAUSE	Module number is valid, but corresponding module is not installed in the system.

HEX	`cea0 n`
CONSOLE	`***Invalid device adapter number n`
CAUSE	Device adapter number was not between 0 and 15 inclusive.

HEX	`cea8`
CONSOLE	`***No tests specified`
CAUSE	At least one test mode must be specified.

HEX	`cea9`
CONSOLE	`***Number out of range`
CAUSE	The number entered is to large or to small to be accepted.

HEX	`ceaa [bc] ca da`
CONSOLE	`***On [BC bc,] CA ca, device adapter da failed selftest`
CAUSE	The DA sense register PST bit is not set, indicating the DA selftest failed last time it was run.

HEX	`cead n`
CONSOLE	`***Invalid Bus Converter number n`
CAUSE	Bus Converter module number outside the range 1 to 63 was entered.

HEX	`ceaf n`
CONSOLE	`***Module n does not exist`
CAUSE	Module n does not exist on HP-PB.

HEX	`ceb4 [bc] ca da`
CONSOLE	`***On [BC bc], CA ca, device adapter da does not exist`
CAUSE	The specified da number does not correspond with a da installed on the specified ca.

HEX	`ceb5 [bc] ca da`
CONSOLE	`***On [BC bc], CA ca, device adapter da is not an HP-IB DA`
CAUSE	The specified da number does not correspond with a da installed on the specified ca.

HEX	`cebe n`
CONSOLE	`***Maximum error count n exceeded; program will abort`
CAUSE	The maximum error count *n* has been exceeded. Errors associated with the current loop are displayed, then the program returns to ISL.

HEX	`cec4 stat`
CONSOLE	`***Returned info failed with status = stat; cannot selftest`
CAUSE	Entry test failed when called with return info option and the status returned by the Entry test was *stat*.

HEX	`cec5 stat`
CONSOLE	`***Execute step failed with status = stat`
CAUSE	Entry test failed when called with execute step option and the status returned by the Entry test was *stat*.

HEX	`cec6 stat`
CONSOLE	`***Return message failed with status = stat`
CAUSE	Entry test failed when called with return msg option and the status returned by the Entry test was *stat*.

HEX	`cec7 stat`
CONSOLE	`***Describe section failed with status = stat`
CAUSE	Entry test failed when called with describe section option and status returned by the Entry test was *stat*.

HEX	`cec8`
CONSOLE	`***Busy write master register failed on A1126A Bus Converter board`
CAUSE	Busy write register failed on the BC card.

HEX	`cec9`
CONSOLE	`***Busy read master register failed on A1126A Bus Converter board`
CAUSE	Busy read register failed on the BC card.

HEX	`cecA`
CONSOLE	`***Path selected is not a valid module`
CAUSE	The path selected for the diagnostic is not valid.

HEX	`ced1`
CONSOLE	`***Pending register 0 on the A1126A Bus Converter HP-PB board failed`
CAUSE	The pending register 0 on the BC card failed.

HEX	`ced2`
CONSOLE	`***Pending register 1 failed on the A1126A Bus Converter HP-PB board`
CAUSE	The pending register 1 on the BC card failed.

HEX	`ced3`
CONSOLE	`***Pending register 2 failed on the A1126A Bus Converter HP-PB board`
CAUSE	The pending register 2 on the BC card failed.

HEX	`ced4`
CONSOLE	`***Pending register 3 failed on the A1126A Bus Converter HP-PB board`
CAUSE	The pending register 3 on the BC card failed.

HEX	ced5
CONSOLE	***The BC is losing transactions
CAUSE	The BC is losing transactions, the problem cannot be isolated to a card.

HEX	ced6
CONSOLE	***Cannot find A1126A Bus Converter HP-PB port. Probable causes: a) Bad/No cable connection to A1126A Box or b) No power on A1126A box or c) Bad A1126A Bus Converter HP-PB board or d) Bad A1126A Bus Converter Midbus board
CAUSE	Bad or no cable connection to A1126A, or no power on A1126A, or bad Remote card, or bad Mid-Bus card.

HEX	ced8 estat
CONSOLE	***A1126A Bus Converter HP-PB board should have turned off and no slave acked HP-PB transactions. For some reason it registered error. ESTAT = n
CAUSE	The BC card has failed and refuses to turn itself off.

HEX	ced9
CONSOLE	***Either the A1126A Bus Converter HP-PB board slave acked when it was not suppose to or the I/O card on HP-PB is bad
CAUSE	The MAP/PSI card could be the problem, or the BC remote card.

HEX	`ceda stat`
CONSOLE	`***Could not read ENTRY TEST from IODC ROM on A1126A Bus Converter. PDC returned status = stat`
CAUSE	The ROM on the BC Mid-Bus card could be the problem.

HEX	`cedb`
CONSOLE	`***850/855 BC should have pulled error on Midbus and A1126A Bus Converter Midbus board should have gone into DIN mode. Either 850/855 BC or the A1126A Midbus Bus Converter board failed`
CAUSE	Error can be isolated to the 9000/850/855 Bus Converter or the A1126A Mid-Bus card.

HEX	`cedc`
CONSOLE	`***A1126A Bus Converter Mid-Bus board Io error Sadd register did not correctly register the slave address`
CAUSE	The Io error Sadd register on the A1126A Mid-Bus card failed.

HEX	`cedd`
CONSOLE	`***A1126A Bus Converter Mid-Bus board Io error Madd register did not correctly register the slave address`
CAUSE	The Io error Madd register on the A1126A Mid-Bus card failed.

HEX	`cede`
CONSOLE	`***Power warn bit on the A1126A Bus Converter Mid-Bus Io status register did not register power fail`
CAUSE	Probable cause, the power warn bit on the A1126A Mid-Bus card.

HEX	`cedf`
CONSOLE	`***Power fail bit on the A1126A Bus Converter Mid-Bus Io status register did not register power fail`
CAUSE	Probable cause, the power fail bit on the A1126A Mid-Bus card.

HEX	`cee0`
CONSOLE	`***Power lost bit on the A1126A Bus Converter Mid-Bus Io status register did not register power fail`
CAUSE	Probable cause, the power lost bit on the A1126A Mid-Bus card.

HEX	`cee1`
CONSOLE	`***A1126A Bus Converter does not synthesize power fail interrupts correctly fail`
CAUSE	If the BC Mid-Bus port received power fail info as indicated by the status register, it did not generate a broadcast to the external interrupt register on the processor.

HEX	`cee2`
CONSOLE	`***Data parity error in A1126A Bus Converter Mid-Bus port`
CAUSE	Mid-Bus port register data parity error.

HEX	`cee3`
CONSOLE	`***A1126A Bus Converter Mid-Bus port did not find a slave at the address it was trying to access on the Mid-Bus`
CAUSE	Mid-Bus port was probably trying to access a memory location that does not exist.

HEX	`cee5`
CONSOLE	`***Error caused by A1126A Bus Converter Mid-Bus port trying to read A1126A Bus Converter ports registers`
CAUSE	When the Mid-Bus port was trying to read a HP-PB ports register, it encountered an error.

HEX	`cee6`
CONSOLE	`***A1126A Bus Converter Mid-Bus port could not accees A1126A Bus Converter HP-PB ports registers`
CAUSE	The high speed cable or the BC power could be the problem.

HEX	`cee7`
CONSOLE	`***Bad link parity detected by A1126A Bus Converter Mid-Bus port`
CAUSE	Replace the high speed cable.

HEX	`cee8`
CONSOLE	`***A1126A Bus Converter Mid-Bus port as slave, detected Processor or 805/855 BC pull error signal on Mid-Bus`
CAUSE	When the processor (or 850/855 Bus Converter) was accessing the A1126A Mid-Bus port, they detected an error.

HEX	`cee9`
CONSOLE	`***Data parity error detected by A1126A Bus Converter HP-PB port when HP-PB I/O was reading or writing from/to Mid-Bus`
CAUSE	Check the MAP/PSI card and the HP-PB port of A1126A BC.

HEX	`ceea`
CONSOLE	`***A1126A Bus Converter HP-PB port detected address parity error when HP-PB I/O card was reading or writing to Mid-Bus`
CAUSE	Check the MAP/PSI card and the HP-PB port of A1126A BC.

HEX	`ceeb`
CONSOLE	`***A1126A Bus Converter HP-PB port detected error on HP-PB possibly during a read or write to its registers`
CAUSE	Check the HP-PB port of the A1126A or the MAP/PSI card.

HEX	`ceec`
CONSOLE	`***Error while accessing A1126A Bus Converter HP-PB ports registers`
CAUSE	Check the high speed cable, power, and the HP-PB port.

HEX	`ceed`
CONSOLE	`***Bad link parity detected by A1126A Bus Converter port`
CAUSE	Check the high speed cable.

HEX	`ceee`
CONSOLE	`***A1126A Bus Converter HP-PB port as slave detected an HP-PB I/O card assert error signal`
CAUSE	Check the MAP/PSI card.

HEX	cf01
CONSOLE	***Either no type B DMA cards are plugged to HP-PB bus or A1126A Bus Converter HP-PB port cannot access it
CAUSE	A MAP/PSI card needs to be installed for diagnostics, or check for proper insertion of existing cards.

HEX	cf02
CONSOLE	***Warn: Module is not a A1126A Bus Converter. Could be a 850/855 Bus Converter
CAUSE	Indicates that a Bus Converter was found, but it is not an A1126A.

HEX	cf03
CONSOLE	***Module is not a Bus Converter
CAUSE	The module being looked at is not an A1126A Bus Converter.

HEX	cf04
CONSOLE	***The diagnostic could not find any A1126A Bus Converters If one is plugged in then the probable causes are a) Bad ROM on A1126A Bus Converter Midbus board or b) Bad A1126A Bus Converter Midbus board
CAUSE	The diagnostic could not talk to the A1126A Bus Converter.

Contents

Tables

9

PCX Uni-Processor Diagnostic (UNIPROC)

Introduction

The UNIPROC diagnostic is intended to provide factory and field personnel with a tool to quickly test and diagnose the in-system function of the PCX board(s).

Defects And Enhancements

Submit defect reports and enhancement requests concerning this diagnostic through the STARS database, referencing product number 30343-10005.

Minimum Configuration

The following minimum configuration is required to be able to run this diagnostic:

- An HP 9000 870/980 SPU
- A system console
- A boot disk or tape
- ISL and working PDC

Diagnostic Organization

The diagnostic consists of 122 test sections and a control program which provide a means to control execution order and interfacing to common procedures provided by the user interface module. The diagnostic sections and the functions tested by each test are listed below. The significance of the section is that it is the smallest looping entity provided. Looping can be done on any set of sections, including all or any one of them.

SECTION	AREA TESTED
1	CPU general registers, diagnose functions
2	CPU ALU functions (arithmetic and logical)
3	CPU Shifter and Shift Amount register
4	CPU arithmetic conditions
5	CPU PSW Carry/Borrow bits
6	CPU control registers, as well as space registers
7	PMI registers, pattern tests for both SPI and PMIN registers such as: FLEX, STATUS, etc.
8	PMI tests for I/O register accesses, interrupt mechanism, and TMOUT reset and increment (SilverBullet does Timeout and reset in HPMC test)
9	PMI tests for more of the STATUS bits (Panther's NRM, MOP, T-Lock, and TMO, etc., as well as tombstone registers, LPMC); (SilverBullet tests all the status bits except PON)
10	PMI extended tests (Panther: Read Halfline, Clear, and Copyout functions); (SilverBullet: LDCW)
11	ICMUX Diagnose commands and diagnose registers
12	ICMUX RAM Array pattern tests for Set 1
13	ICMUX Ram Array pattern tests for Set 2
14	ICMUX cache addressing and hashing test
15	ICMUX Parity and error detection
16	ICMUX RPN registers and comparators
17	ICMUX Flush functionality
18	Secondary I-TLB ram array pattern test
19	Secondary I-TLB addressing and hash test
20	Secondary I-TLB minimum functionality (purge and insert) test
21	Secondary I-TLB parity generation and detection circuits
22	DCMUX Diagnose commands and registers
23	DCMUX Ram Array pattern tests for Set 1
24	DCMUX Ram Array pattern tests for Set 2
25	DCMUX Cache addressing and hashing test

26	DCMUX basic functionality
27	DCMUX Parity, error detection, and single bit error correction
28	DCMUX Load and Store operations
29	DCMUX (DATA CACHE) write buffer test
30	Secondary D-TLB ram array pattern test
31	Secondary D-TLB addressing and hash test
32	Secondary D-TLB minimum functionality (purge and insert) test
33	Secondary D-TLB parity generation and detection circuits
34	TLB RAM (on chip I and D) array
35	TLB (on chip I and D) lock bit functionality
36	TLB (on chip I and D) error detection
37	TLB (on chip I and D) access rights and, access ID checking
38	TLB (on chip I and D) Trap bits functionality (B,D,T)
39	Virtual data translation
40	Access Rights functionality
41	Basic instructions; branch (BL), compare immediate and branch (COMIBF and COMIBT) with short displacements, and add immediate (ADDI)
42	Real addressing mode branching with a range of 512kb using the BL instruction with gr0 as the link register
43	The compare and branch instructions (COMBT, COMBF) with all possible condition fields
44	Simple test of load offset (LDO) instruction
45	Real addressing mode branching with range of 256kb using the branch vectored (BV) instruction
46	Real addressing mode loads using LDW, LDWM and LDBS instruction. Also simple tests of LDIL and ADD.
47	Basic functionality of the interrupt handling instructions (IITLBA, IITLBP). Also test move to control instruction (MTCTL) using the IVA, PCQS,PCOQ and IPSW registers.
48	Memory control instructions (IITLBA, IITLBP, PITLB), with absolute loads and stores (LDWAX, LDWAS and STWAS).
49	Memory control instructions (IDTLBA, IDTLBP, PDTLB, PROBER, PROBEW, PROBERI, and PROEWI) and absolute loads and store instructions (LDWAX, LDWAS, STWAS, and LPA).
50	Virtual mode branching using B and BV.
51	Variable extract instructions (VEXTRU,VEXTRS) with all condition codes.
52	Variable deposit instruction (VDEP) with all condition codes.

53	ADD, ADDO, ADDC, ADDCO, and ADDL instructions, conditional nullification and value of PSW carry/borrow bits. Tests overflow trap for ADDO, SUBB and SUBTO and trap on condition for ADDT and SUBTO.
54	ADDI, ADDIO, ADDIT and ADDITO instructions, conditional nullification and value of PSW carry/borrow bits. Tests overflow trap for ADDIO and ADDITO, and trap on condition for ADDIT and ADDITO.
55	AND instruction with conditional nullification.
56	The compare and clear instruction (COMCLR).
57	OR instruction with conditional nullification.
58	SUB, SUBO, SUBB, SUBBO, SUBT, and SUBTO instructions, conditional nullification, and value of PSW carry/borrow bits. Tests overflow trap for SUBO, SUBB and SUBTO, and trap on condition for SUBT and SUBTO.
59	UXOR instruction with conditional nullification.
60	XOR instruction with conditional nullification.
61	Add and branch instructions (ADDBT, ADDBF, ADDIBT, and ADDIBF) with conditional nullification.
62	Add immediate left (ADDIL) and Load immediate left (LDIL) instructions.
63	And complement instruction (ANDCM) with conditional nullification.
64	Deposit instructions (DEPI, VDEPI, DEP, ZDEP, ZDEPI, ZVDEP and ZVDEPI) with all condition codes.
65	Extract instructions (EXTRU and EXTRS).
66	Move and Branch instruction (MOVB) with conditional branching and nullification.
67	Subtract from immediate instructions (SUBI and SUBIO).
68	Unit add complement instructions (UADDCM and UADDCMT).
69	Taken Branch trap on all conditional and unconditional branch instructions (BL, BLR, BV, BE, BLE and GATE).
70	Compare immediate and clear instruction (COMICLR).
71	Decimal correction instructions (DCOR and IDCOR). Also performs complete test of PSW carry/borrow bits.
72	Shift and add instructions (SH1ADD, SH1ADDO, SH1ADDL, SH2ADD, SH2ADDO, SH2ADDL, SH3ADD, SH3ADDO and SH3ADDL). Also tests overflow traps.
73	Shift double instructions (SHD and VSHD).
74	Absolute addressing load and store instructions (LDWAX, LDWAS and STWAS) including register modification, pre- and post- incrementing.
75	Data access break mechanism including the PSW x bit and the b bit in the TLB entry.
76	The TLB dirty bit update trap.

77 Privilege traps at privilege levels less than the highest for the following instructions (GATE, RFI, SSM, RSM, LPA, IDTLBA, IDTLBP, IITLBA, IITLBP, PITLB, PDTLB, MTSP, MTCTL and MFCTL).

78 Memory protection id validation for data accesses.

79 Memory protection id validation for code accesses.

80 Branch on bit instructions (BB and BVB) for different bit positions.

81 Divide step instruction (DS).

82 Load offset instruction (LDO).

83 Load and store byte instructions (LDB, LDBS, LDBX, STB and STBS) with address generation, register modification and pre- and post- increment.

84 Load and store word instructions (LDWM, LDWX, LDWS, STWM and STWS) with address generation, register modification, pre- and post-increment. Also tests sr's 1-3.

85 Load and store half word instructions (LDH, LDHS, LDHX, STH and STHS) with address generation, register modification, and pre- and post- increment.

86 Store byte short instruction (STBYS) with address generation, register modification, and pre- and post increment. Uses different spaces for source and destination.

87 Load and clear word instructions (LDCWX and LDCWS) with address generation, register modification, and pre- and post- increment.

88 Virtual address branching using BL, BLR, BV, BE, BLE, and GATE instructions.

89 Tests that instructions which should set PSW carry/borrow bits do set them, and instructions which shouldn't set them, don't.

90 Extended tests of the Add and Subtract instructions with and without nullification under various conditions and with various registers.

91 Extended tests of the AND, OR, XOR, and ANDCM instructions with and without nullification under all possible conditions.

92 Extended tests of the Shift, Extract and Deposit instructions with and without nullification under conditions generated. Tests each instruction with varying registers, position and length.

93 Extended tests of the Loads and Store instructions with emphasis on consistency in address calculation.

94 Extended tests of the Branch instructions (conditional and unconditional) with and without nullification, and with random displacements.

95 Extended tests of Overflow and Condition traps caused by Add, Shift and Add, and Subtract instructions, with and without nullification.

96 Extended tests of the Higher/Lower Privilege Transfer, Privileged Register, and Privileged Operation traps.

97 Extended tests of the Instruction/Data Memory Protection, Data Memory Break, Instruction/Data TLB Miss Fault, TLB Dirty Fault, and Non-Access

	Data TLB Miss Fault traps. Instructions tested include IDTLBA, IDTLBP, IITLBA, IITLBP, PDTLB, PITLB, PROBER, PROBERI, PROBEW, PROBEWI, and LPA.
98	Extended tests of the unit instructions, UXOR, UADDCM, DCOR, IDCOR and DS, with and without nullification.
99	Loads and stores to and from the FPC floating point registers and the ability to copy from one to another.
100	Ability to convert operands in the floating point registers to absolute form, i.e. positive.
101	Ability to detect attempted use of reserved registers or operations.
102	Ability of ADD/SUB chip to do single precision floating point addition.
103	Ability of ADD/SUB chip to do double precision floating point addition.
104	Ability of ADD/SUB chip to do single and double precision floating point subtraction.
105	Less-than floating point comparisons.
106	Equal-to floating point comparisons.
107	Greater-than floating point comparisons.
108	Unordered floating point comparisons.
109	Conversion of single precision floating point operands to double precision.
110	Conversion of double precision floating point operands to single precision.
111	Conversion of single precision floating point operands to single and double precision fixed point format.
112	Conversion of double precision floating point operands to single and double precision fixed point format.
113	Conversion of single precision floating point operands to single and double precision fixed point format with the results truncated.
114	Conversion of double precision floating point operands to single and double precision fixed point format with the results truncated.
115	Conversion of single precision fixed point operands to single and double precision floating point format.
116	Conversion of double precision fixed point operands to single and double precision floating point format.
117	Single and double precision floating point multiply.
118	Single and double precision floating point division.
119	Single and double precision floating point round.
120	Single precision instruction queuing and bypassing.
121	Double precision instruction queuing and bypassing.
122	Single & double precision floating point division test.

Diagnostic Coverage

An assessment of the set of tests defined in the preceding list, shows that an attempt is made to test all of the data path hardware, all implemented instructions, and all but one of the traps and interrupts (power fail) are tested for stuck-at faults. The following table indicates the test section which is used to test each functional area within the VLSI chips and the other discrete components on the processor board. Note that certain functional areas cannot be tested directly with this diagnostic. These include the clock drivers, the Revision Port, and the temperature sensor. Malfunction of some of these components may prevent the diagnostic from running, but in all cases some other means must be used for fault isolation.

Table 9-1. Diagnostic Test Coverage

Component	Functional Area	Section# Where Tested	Features Untested
CPU	General Registers	1	
	Control Registers	6	
	Instr. Pipe Decode	All	
	Data Out Registers	48,49	
	Program Counter Pipe	All	
	Branch Adder	42,45	
	ALU	2,55,57,59,60	
	Mask/Merge Shifter	51,52,64,65,71,72	
	Result Register	All	
	Trap Logic	8,53,54,58,69,71, 75/79,93,94,96	Power-On Trap
	Instr Decoder	All	
TLB(CPU)	Protection Stack	40	
	Address Stack	19,31	
	Hashing	19,31	
	TLB Tag	34	
	Control Logic	20,32,35/39	
TLB	Tag RAMs	34	
	RPN RAMs	34	
ICMUX	RPN/TAG Stack	16	
	Data Stack	11	
	Control Logic	15,17,20,21	
	(CPU) Address Stack	14	
	(CPU) Hashing	14	
	(CPU) ICCU Control Logic	15,17	
Set 1	Tag/Data RAMs	12,14,18	
Set 2	Tag/Data RAMs	13,14,18	

Table 9-1. Diagnostic Test Coverage (continued)

Component	Functional Area	Section# Where Tested	Features Untested
DCMUX	RPN/TAG Stack	29	
	Data Stack	22,29	
	Control Logic	26/29,32,33	
	(CPU) Address Stack	25	
	(CPU) Hashing	25	
	(CPU) DCCU Control Logic	26/29	
Set 1	Tag/Data RAMs	23,25,30	
Set 2	Tag/Data RAMs	24,25,30	
PMI	Address Stack	8	
	Status Stack	9	
	Control Stack	7	
	Buffer Stack	9	
FPC	FP Registers	99,100	
	Status Register	99,100	
	Operation/Excep Reg	99,100	
	Control Logic	99,100,101,120,121	
FALU	Add/Sub Pipe	102/116, 119	
FMUL	Multiply/Divide Pipe	117	

Diagnostic Operation

UNIPROC operation assumes that the system will include a disk or tape and a system console. When power is applied to the system, the PDC will initialize each processor and then cause them to execute the internal hardware self-test. If these pass, then the monarch processor will boot the system, which will cause ISL to be loaded and launched. This in turn will result in the ISL prompt ISL> appearing on the system console. At this point, the user can direct the system to load any one of several programs, such as the operating system or the Processor Offline Diagnostic. The latter is selected by typing uniproc. The entire diagnostic is then loaded into memory and control is transferred to the starting point. The following banner is then displayed on the console:

```
Series AXXXXAP System Processor Unit Diagnostic, Version 1.0
There are 121 sections in this diagnostic - 1 thru 121
```

This is followed by the diagnostic prompt UNIPROC> and the system then waits for user commands. At this point, the user can select which processors to test, which sections to run, whether to enable activity printouts, error and isolation messages, error or isolation pauses, or enable looping. Hardcopy printout can also be selected at this point. If no changes are made the defaults are: all processors, all test sections, activity indicators enabled, error and isolation messages enabled, pause after isolation message enabled, and no looping or hardcopy. To begin execution, it is only necessary to type RESUME. Note that typing RUN UNIPROC will also work; RUN will always start from the beginning of the tests selected. The first message to appear after the RESUME command is given, assuming processor 0 is enabled, will be:

```
DIAGNOSTIC STARTED ON PROCESSOR #0
```

If activity printouts have been selected, then the following display will be observed as the diagnostic progresses through each section (assuming all sections have been selected):

```
STARTING CPU DATA PATH TESTS - SECTIONS 1/6
SECTION 001
SECTION 002
SECTION 003
SECTION 004
SECTION 005
SECTION 006
STARTING PMI DATA PATH TESTS - SECTIONS 7/10
SECTION 007
SECTION 008
SECTION 009
SECTION 010
(etc)
```

The following message will be displayed after the last test has been executed:

```
DIAGNOSTIC COMPLETED ON PROCESSOR #0
```

If more than one processor is installed, then the preceding message sequence will be repeated for each processor selected.

If looping has been enabled, then at the end of each pass through the diagnostic, a message of the following form will be displayed:

```
END OF PASS 0000000001
```

If error messages have been enabled and a fault is detected, then a message of the following form will be displayed:

```
Error no. 003 Detected in Section 012
```

Error numbers range from a few to over 100 per section, starting with 1. If isolation messages have been enabled, then immediately following the error message will appear a message of the form:

```
THE FAULT IS MOST LIKELY IN THE CPU CHIP
```

In some cases, additional components may be designated if the symptom is such that the fault cannot be resolved to only one chip. The exact function that failed can be determined by using the section and error numbers to index into the diagnostic listing.

When there is more than one processor installed, and the user wants to test only one, then the `proc n` command will allow him to select which one to test. Typing `RESUME` will then cause that processor to run the diagnostic.

The following shows the screen output after the user calls UNIPROC from the ISL prompt, and reaches the diagnostic prompt:

```
UNIPROC> resu

DIAGNOSTIC STARTED ON PROCESSOR #0

STARTING CPU DATA PATH TESTS - SECTIONS 1/6
SECTION 001
SECTION 002
SECTION 003
SECTION 004
SECTION 005
SECTION 006
STARTING PMI DATA PATH TESTS - SECTIONS 7/10
SECTION 007
SECTION 008
SECTION 009
SECTION 010

ERROR NO. 003 DETECTED IN SECTION 010

THE FAULT IS MOST LIKELY IN THE PMI CHIP

UNIPROC> resu

STARTING I-CMUX DATA PATH TESTS - SECTIONS 11/21
SECTION 011
SECTION 012
   .
   .
   .
SECTION 121

DIAGNOSTIC COMPLETED ON PROCESSOR #0

UNIPROC>
```

Operational Commands

A variety of commands are provided via the user interface to enable the user to run the diagnostic in the manner he wants. Those that are relevant to the processor diagnostic are briefly summarized below. The default values for the run time options that these commands affect, are also listed in the table. It is only necessary to type the first four letters of each command. Also, more than one command can be typed on the same line by separating them with semicolons.

Table 9-2. Operational Commands

COMMAND	FUNCTION	DEFAULTS
changeio	Change paths to boot devices, console or printer	
creg	Display all control registers at end of last test	
debug	Invoke the Software Maintenance Monitor (MON)	
eepr	Enable error message printing	on
eeps	Enable a pause after an error is detected	
eipr	Enable isolation message printing	on
eips	Enable a pause after an isolation message is printed	on
enpr	Enable non-error message printing	on
enps	enable a pause after a non-error message is printed	on
erro	Shorthand for eepr;sipr;snpr	
errp	Shorthand for epps;sips;snps	
exit	Return to ISL	
freg	Display current contents of the floating point registers	
hard	Force all messages to a hardcopy device	off
help	Print information on operating the diagnostic	
info	print information about diagnostic (e.g. what sections test what)	
listio	List the I/O paths to boot devices, console and printer	
loop <n>	Set number of times to loop thru diagnostic, if no n, loop forever	1
proc n	Set numbers of processors to run diagnostic	0/3
pstatus	Display processor chip rev# and various status registers	
registers	Display all general registers at end of last test	
reset	Reinitialize run time options to default values	
resume	Restart diagnostic after a pause	
run uniproc	Start running diagnostic from beginning of current enabled sections	
sect n	Set numbers of test sections to be executed	1/121
sepr	Suppress error message printing	
seps	Suppress a pause after an error is detected	on
sipr	Suppress isolation message printing	

Table 9-2. Operational Commands (continued)

COMMAND	FUNCTION	DEFAULTS
sips	Suppress a pause after an isolation message is printed	
snpr	Suppress non-error message printing	
snps	Suppress a pause after a non-error message is printed	
sstate	Return diagnostic to initial entry point and reinitialize (start state)	
state	Display the current values of run time options	
stop	Stop ISL from continuing the auto boot sequence	

Error Messages

The following is a list of the error messages which are generated by this diagnostic.

SECTION 001:

ERROR NO	MEANING
1	General register 0 error
2	All's error
3	All 0's error
4	Alternating 1's error
5	Alternating 0's error
6	Crosstalk error
7	space reg1 stuck-at
8	space reg2 stuck-at
9	space reg3 stuck-at
10	space reg0 stuck-at
11	space reg4 stuck-at
12	space reg5 stuck-at
13	space reg6 stuck-at
14	space reg7 stuck-at
15	CPU diagnose register err
16	space reg1 bad access
17	space reg2 bad access
18	space reg3 bad access
19	space reg4 bad access
20	space reg5 bad access
21	space reg6 bad access
22	space reg7 bad access

SECTION 002:

ERROR NO	MEANING
1	Combf/combt error
2	Add/Sub error
3	Addi/Subi error
4	And/Or/Xor/Uxor error
5	SHxAdd error
6	Uaddcm error
7	Andcm error
8	Worst case carry error

SECTION 003:

ERROR NO	MEANING
1	Shift by 1 error
2	Shift by 2 error
3	Shift by 4 error
4	Shift by 8 error
5	Shift by 16 error
6	Shift by 31 error
7	SAR error (cr11)

SECTION 004:

ERROR NO	MEANING
1	never error
2	overflow error
3	odd error
4	unit true error
5	unit false error
6	arith = error
7	arith < error
8	arith <= error
9	zero error
10	overflow error
11	never error
12	overflow error
13	odd error
14	unit true error
15	unit false error
16	arith = error
17	
18	arith <= error
19	zero error
20	overflow error
21	never error
22	overflow error
23	odd error
24	unit true error
25	unit false error
26	arith = error

SECTION 005:

ERROR NO	MEANING
1	Carry/borrow error
2	Carry/borrow error
3	Carry/borrow error
4	Carry/borrow error
5	Carry/borrow error
6	Carry/borrow error
7	Carry/borrow error
8	Carry/borrow error
9	Carry/borrow error
10	Carry/borrow error
11	Carry/borrow error
12	Carry/borrow error
13	failed to set mask[27-30]
14	failed to set mask[28,30]
15	failed to set mask[29-31]
16	stuck mask[27-31] bits
17	stuck mask[27-31] bits
18	stuck mask[27-31] bits
19	stuck mask bit 31
20	stuck mask bit 30
21	stuck mask bit 29
22	stuck mask bit 28
23	stuck mask bit 27
24	stuck mask bit 28
25	stuck mask bit 29
26	stuck mask bit 30
27	stuck mask bit 31

SECTION 006:

ERROR NO	MEANING
1	cr10 error
2	cr0 error
3	cr14 error
4	cr0 counting error
5	cr18 error
6	cr18 error
7	cr10 error
8	cr19 error
9	cr19 error
10	cr22 error
11	cr18 error
12	cr18 error
13	cr17 error
14	cr17 error
15	cr15 error
16	cr24 error
17	cr25 error
18	cr26 error
19	cr27 error
20	cr28 error
21	cr29 error
22	cr30 error
23	cr31 error
24	eir error
25	itlim error
26	itcnt error
27	cr24 error
28	cr25 error
29	cr26 error
30	cr27 error
31	cr28 error
32	cr29 error
33	cr30 error
34	cr31 error
35	cr8;cr9;cr12;cr13 error
36	interval timer error
37	interval timer error
38	interval timer error
39	interval timer error
40	interval timer error
41	interval timer error
42	cr22 error
43	cr22 error
44	cr22 error
45	cr22 error
46	cr22 error

SECTION 007:

ERROR NO	MEANING
1	tmout error
2	flex error
3	stat error [8:16]
4	stat error [20:28]
5	stat error, can't clear flags
6	TLBSID0 error
7	TLBVPN0 error
8	Flush buffer memory error
9	Flush buffer error
10	TLBSID0 error
11	TLBVPN0 error
16	PBAD1 error
17	CCCAD0 error
18	CCCCMD0 error
19	IFAD0 error
20	PFAD0 error
21	Extended Flush buffer error
22	Extended Flush buffer error
23	prefetch error
24	strange hpmc in ext ifad0 test
25	IFAD0 error
26	IFAD0 error
27	address range error
28	address range error for flush extended buffer test
29	hpmc on test addr pattern in extended buffer test
30	hpmc on test addr pattern in extended buffer test
31	hpmc on test addr pattern in extended buffer test
32	hpmc on test addr pattern in extended buffer test

SECTION 008:

ERROR NO	MEANING
1	PMI rev code error
2	PMI rev code error
3	IO_EIR set error
4	EIR set error
5	Interval timer interrupt failed
6	EIR reset error
7	Read Shared error
8	Read Private error
9	Read Private error
10	Purge cache erro
11	Tmout reset error
12	Tmout not reset error
13	Tmout increment error
14	External interrupt occurred
15	External interrupt didn't occur
16	HPMC did occur
17	HPMC did not occur
18	TOC did not occur
19	TOC did occur

SECTION 009:

ERROR NO	MEANING
1	HPMC didn't occur
2	NRM bit wasn't set
3	TMO bit wasn't set
4	HPMC didn't occur
5	NRM bit wasn't set
6	TMO bit wasn't set
7	HPMC didn't occur
8	MER bit error
9	DAP bit error
10	wrong vi bits error
11	tombstone error
12	tombstone error
13	tombstone error
14	fix field error
15	wrong vi bits error
16	pmi failed to cause lpmc
17	tombstone error
18	tombstone error
19	tombstone error
20	tombstone error
21	tombstone error
22	tombstone error
23	tombstone error
24	tombstone error

SECTION 010:

ERROR NO	MEANING
1	Register error
2	Crosstalk error
3	Interval timer error
4	Rev error
5	tmout did not reset?
6	load & clear error
7	load & clear error
8	load & clear error
9	load & clear error
10	load & clear error
11	load & clear error
12	load & clear error

SECTION 11:

ERROR NO.	MEANING
1	version number error
2	reg.(1-9) did not match expected pattern
3	DPE bit did not update on a parity error
4	freeze bit was not set on a parity error
5	ILPMC_BIT did not update correctly for a parity error.

SECTION 12:

SAMPLE ERROR MESSAGE

```
I-CACHE RAM ERROR IN SET0 AT ADDRESS =00000

Expected: Data=00000000, Par=0, Tag=00000000, Par=0

Actual:   Data=00000000, Par=0, Tag=00000000, Par=0
```

INTERPRETATION OF ERROR MESSAGE:

The parameters (Data,Par(data parity), Tag, Par(tag parity)) in the previous displayed error message maps to the data input lines of the rams in the following manner:

Data[0:31] = G1DL[0:31]/G0DL[0:31], Par[0:3] = G0DCL[0:3]/G1DCL[0:3]

Tag[5:26] = G0TL[0:21], Par[2:3] = G0TCL[4:5]

Isolation to Signal For SET0 Errors

ADDRESS[19]	Data/Par	Tag/Par	SIGNAL
X	Actual=Expected	Actual<>Expected	G0TL[0:21]/G0TCL[4:5]
0	Actual<>Expected	Actual=Expected	G0DL[0:31]/G0DCL[0:3]
1	Actual<>Expected	Actual=Expected	G1DL[0:31]/G1DCL[0:3]

SECTION 13:

SAMPLE ERROR MESSAGE:

```
I-CACHE RAM ERROR IN SET1 AT ADDRESS =00000

Expected: Data=00000000, Par=0, Tag=00000000, Par=0

Actual:   Data=00000000, Par=0, Tag=00000000, Par=0
```

INTERPRETATION OF ERROR MESSAGE:

The parameters (Data,Par(data parity), Tag, Par(tag parity)) in the previous displayed error message maps to the data input lines of the rams in the following manner:

Data[0:31] = G1DL[0:31]/G0DL[0:31], Par[0:3] = G0DCL[0:3]/G1DCL[0:3]

Tag[5:26] = G1TL[0:25], Par[2:3] = G1TCL[4:5]

Isolation to Signal For SET1 Errors

ADDRESS[19]	Data/Par	Tag/Par	SIGNAL
X	Actual=Expected	Actual<>Expected	G1TL[0:25]/G1TCL[4:5]
0	Actual<>Expected	Actual=Expected	G1DL[0:31]/G1DCL[0:3]
1	Actual<>Expected	Actual=Expected	G0DL[0:31]/G0DCL[0:3]

SECTION 14:

ERROR NO.	MEANING
1	I-cache tag addressing error
2	I-cache tag parity addressing error
3	I-cache check bits addressing error
4	I-cache data addressing error
5	I-cache hashing error

SECTION 15:

ERROR NO.	MEANING
1	instruction not executed on a cache miss
2	LPMC trap taken on cache miss
3	instruction not executed after a cache tag LPMC (tag parity bit error).
4	LPMC not taken on a tag parity bit error.
5	error in syndrome generation, syndrome should be bad.
6	error in syndrome generation, syndrome should be good.
7	second level TLB error signaled on a TAG parity error
8	LPMC bit not set on TAG parity bit error.
9	instruction not executed after a cache tag LPMC (tag data bit error)
10	LPMC not taken on a tag data bit error.
11	error in syndrome generation, syndrome should be bad.
12	error in syndrome generation, syndrome should be good.
13	second level TLB error signaled on a TAG data bit error
14	LPMC bit not set on TAG data bit error.
15	instruction not executed after a I-cache data LPMC (data parity bit error).
16	LPMC not taken on a data parity bit error.
17	error in syndrome generation, syndrome should be bad.
18	error in syndrome generation, syndrome should be good.
19	second level TLB error signaled on a data parity error
20	LPMC bit not set on data parity bit error.
21	instruction not executed after a I-cache data LPMC (data bit error)
22	LPMC not taken on a data bit error.
23	error in syndrome generation, syndrome should be bad.
24	error in syndrome generation, syndrome should be good.
25	second level TLB error signaled on a data bit error
26	LPMC bit not set on data bit error.

SECTION 16:

ERROR NO.	MEANING
1	Virtual instruction mode error in RPN comparators
2	HPMC occurred for virtual mode RPN comparator test
3	real instruction mode error in RPN comparators
4	HPMC occurred for real mode RPN comparator test

SECTION 17:

ERROR NO.	MEANING
1	fic to set1 didn't flush
2	fic to set2 did flush
3	fic to set2 didn't flush
4	fic to set1 did flush
5	fice to set1 didn't flush
6	fice to set2 didn't flush
7	cache line not found
8	line not valid

SECTION 18:

SAMPLE ERROR MESSAGE:

```
SECOND LEVEL I-TLB RAM ERROR AT ADDRESS = 0000

Expected:   wd0/wd1=00000000, wd2/wd3=00000000, goc/gic=00000000

Actual:     wd0/wd1=00000000, wd2/wd3=00000000, goc/gic=00000000
```

INTERPRETATION OF ERROR MESSAGE:

The parameters (wd0/wd1, wd2/wd3, goc/gic) in the previous displayed error message maps to the data input lines of the rams in the following manner:

wd0/wd1[5:26] = G0TL[0:21], goc[9:10] = G0TCL[4:5]

wd2/dw3[5:30] = G1TL[0:25], gic[15:16] = G1TCL[4:5]

Isolation to Signal For Second Level I-TLB Errors

wd0/wd1	wd2/wd3	goc	gic	SIGNAL
error	---	---	---	G0TL[0:21]
---	---	error	---	G0TCL[4:5]
---	error	---	---	G1TL[0:25]
---	---	---	error	G1TCL[4:5]

SECTION 19:

ERROR NO.	MEANING
1	I-tlb2 tag0 addressing error
2	I-tlb2 tag1 addressing error
3	I-tlb2 check bits addressing error
4	I-tlb2 hashing error

SECTION 20:

ERROR NO.	MEANING
1	L2 TLB error ;trap occurred or result_val not updated
2	L2 insert TLB error. entry not inserted correctly
3	L2 purge TLB error ;entry not purged
4	L2 purge TLB error ;entry purge
5	L2 purge entry TLB error ;entry not purged

SECTION 21:

ERROR NO.	MEANING
1	instruction execution failed
2	I-TLB MISS error
3	instruction execution failed
4	I-TLB MISS error
5	incorrect syndrome generation
6	TLB error bit not set
7	ILPMC bit not set

SECTION 22:

ERROR NO.	MEANING
1	odd version number error
2	even version number error
3	odd DCMUX bits of reg.(1-9) did not match expected pattern.
4	even DCMUX bits of reg.(1-9) did not match expected pattern.
5	DPE bit did not update on an odd DCMUX parity error
6	freeze bit was not set on an odd DCMUX parity error
7	DLPMC_BIT did not update for an odd DCMUX parity bit error
8	DPE bit did not update on an even DCMUX parity error
9	freeze bit was not set on an even DCMUX parity error
10	DLPMC_BIT did not update for an even DCMUX parity bit error

SECTION 23:

SAMPLE ERROR MESSAGE:

```
D-CACHE RAM ERROR IN SET0 AT ADDRESS =00000

Expected: Data=00000000 00000000, Ckbits=000, Tag=00000000, Chkbits=00

Actual:   Data=00000000 00000000, Ckbits=000, Tag=00000000, Chkbits=00
```

INTERPRETATION OF ERROR MESSAGE:

The parameters (Data,Ckbits(data parity), Tag, Chkdits(tag parity)) in the previous displayed error message maps to the data input lines of the rams in the following manner:

Even Data Cache Data RAM Array:

Data[0,2,4,6,8..62] = G0DL[0:31], Ckbits[0,2,4,6,8,10] = G0DCL[0:5]

Odd Data Cache Data RAM Array:

Data[1,3,5,7,9..63] = G0DL[0:31], Ckbits[1,3,5,7,9,11] = G0DCL[0:5]

Tag:

Tag[5:28] = G0TL[0:23], Chkbits[2:7] = G0TCL[0:5]

Isolation to Signal For SET0 Errors

ERROR BIT	Data/Par	Tag/Par	SIGNAL	RAM ARRAY
X	Actual=Expected	Actual<>Expected	G0TL[0:23]/G0TCL[0:5]	Tag
Even	Actual<>Expected	Actual=Expected	G0DL[0:31]/G0DCL[0:5]	Even
Odd	Actual<>Expected	Actual=Expected	G0DL[0:31]/G0DCL[0:5]	Odd

SECTION 24:

SAMPLE ERROR MESSAGE:

```
D-CACHE RAM ERROR IN SET1 AT ADDRESS =00000

Expected: Data=00000000 00000000, Ckbits=000, Tag=00000000, Chkbits=00

Actual:   Data=00000000 00000000, Ckbits=000, Tag=00000000, Chkbits=00
```

INTERPRETATION OF ERROR MESSAGE:

The parameters (Data,Ckbits(data parity), Tag, Chkbits(tag parity)) in the previous displayed error message maps to the data input lines of tbe rams in the following manner:

Even Data Cache Data RAM Array:

Data[0,2,4,6,8..62] = G1DL[0:31], Ckbits[0,2,4,6,8,10] = G1DCL[0:5]

Odd Data Cache Data RAM Array:

Data[1,3,5,7,9..63] = G1DL[0:31], Ckbits[1,3,5,7,9,11] = G1DCL[0:5]

Tag:

Tag[5:28] = G1TL[0:25], Chkbits[2:7] = G1TCL[0:5]

Isolation to Signal For SET1 Errors

ERROR BIT	Data/Par	Tag/Par	SIGNAL	RAM ARRAY
X	Actual=Expected	Actual<>Expected	G1TL[0:25]/G1TCL[0:5]	Tag
Even	Actual<>Expected	Actual=Expected	G1DL[0:31]/G1DCL[0:5]	Even
Odd	Actual<>Expected	Actual=Expected	G1DL[0:31]/G1DCL[0:5]	Odd

SECTION 25:

ERROR NO.	MEANING
1	D-cache tag addressing error
2	D-cache tag parity addressing error
3	D-cache check bits addressing error
4	D-cache data addressing error
5	D-cache data hashing error

SECTION 26:

ERROR NO.	MEANING
1	store didn't work
2	incorrect data;store didn't work
3	incorrect rpn_val;store didn't work
4	dirty bit not set for store
5	private_bit not set for store
6	line was not validated (fdc)
7	valid bit not set on store
8	dirty bit not set on store;
9	private bit not set.
10	error;incorrect data
11	error;incorrect tag
12	LDW from set 1 didn't work
13	LDW from set 0 didn't work
14	dirty bit set;LDW error
15	private bit not set;LDW error
16	valid bit not set;LDW error
17	incorrect RPN;LDW error
18	incorrect data;LDW error
19	valid bit not set;FDCE error
20	cache line not invalidated (PDC)
21	cache line was invalidated (PDC)

SECTION 27:

ERROR NO.	MEANING
1	Data not loaded on a D-cache miss
2	LPMC occurred on a D-cache miss
3	fdc with tag parity bit error did not update cache line.
4	LPMC did not occur for FDC with a tag parity bit error.
5	Incorrect syndrome generated for tag parity bit error.
6	bad tag syndrome generated for group not under test.
7	TLB error signaled on a tag parity bit error.
8	D-LPMC bit on cpu0 was not set on D-cache tag parity
9	fdc with tag data bit error did not update cache line.
10	LPMC did not occur for FDC with a tag data parity bit error.
11	Incorrect syndrome generated for tag data bit error.
12	bad tag syndrome generated for group not under test.
13	TLB error signaled on a tag data bit error
14	D-LPMC bit on cpu0 was not set on D-cache tag data bit error.
15	fdc with even data parity bit error did not update cache line.
16	LPMC did not occur for FDC with an even data parity bit error.
17	Incorrect syndrome generated for even data parity bit error.
18	bad CheckBits syndrome generated for group not under test.
19	TLB error signaled on an even data parity bit error.
20	D-LPMC bit on cpu0 was not set on D-cache even data parity bit error.
21	fdc with even data bit error did not update cache line.
22	LPMC did not occur for FDC with an even data bit error.
23	Incorrect syndrome generated for even data bit error.
24	bad data syndrome generated for group not under test.
25	TLB error signaled on an even data bit error.
26	D-LPMC bit on cpu0 was not set on D-cache even data bit error.
27	fdc with odd data parity bit error did not update cache line.
28	LPMC did not occur for FDC with an odd data parity bit error.
29	Incorrect syndrome generated for odd data parity bit error.
30	bad CheckBits syndrome generated for group not under test.
31	TLB error signaled on an even data parity bit error.
32	D-LPMC bit on cpu0 was not set on D-cache odd data parity bit error.
33	fdc with odd data bit error did not update cache line.
34	LPMC did not occur for FDC with an even data bit error.
35	Incorrect syndrome generated for even data bit error.
36	bad data syndrome generated for group not under test.
37	TLB error signaled on an odd data bit error.
38	D-LPMC bit on cpu0 was not set on D-cache odd data bit error.
39	Data compare failed
40	lpmc bit not set for data bit error
41	Data compare failed
42	lpmc bit not set for tag error
43	corrected tag from diag. reg. did not match actual tag.

SECTION 28:

ERROR NO.	MEANING
1	sth op error
2	stb op error
3	stbys,b op error
4	stbys,e op error

SECTION 29:

ERROR NO.	MEANING
1	RPN, RPN bus, CPU error(set 0)
2	RPN, RPN bus, CPU error(set 1)
3	store buffer error (CMUX,CPU)
4	store bypass error (CMUX,CPU)
5	store bypass error (CMUX,CPU)
6	fdc bypass error (CMUX,CPU)
7	pdc bypass error (CMUX,CPU)
8	write buffer interlock error (CMUX,CPU)

SECTION 30:

SAMPLE ERROR MESSAGE:

```
SECOND LEVEL D-TLB RAM HARD FAULT AT ADDRESS = 0000

Expected:   wd0/wd1=00000000, wd2/wd3=00000000, goc/gic=00000000

Actual:     wd0/wd1=00000000, wd2/wd3=00000000, goc/gic=00000000
```

INTERPRETATION OF ERROR MESSAGE:

The parameters (wd0/wd1, wd2/wd3, goc/gic) in the previous displayed error message maps to the data input lines of the rams in the following manner:

wd0/wd1[5:28] = G0TL[0:23], goc[5:10] = G0TCL[0:5]

wd2/dw3[5:30] = G1TL[0:25], gic[11:16] = G1TCL[0:5]

Isolation to Signal For Second Level D-TLB Errors

wd0/wd1	wd2/wd3	goc	gic	SIGNAL
error	---	---	---	G0TL[0:23]
---	---	error	---	G0TCL[0:5]
---	error	---	---	G1TL[0:25]
---	---	---	error	G1TCL[0:5]

SECTION 31:

ERROR NO.	MEANING
1	D-tlb tag0 addressing error
2	D-tlb tag1 addressing error
3	D-tlb check addressing error
4	D-tlb hashing error

SECTION 32:

ERROR NO.	MEANING
1	L2 TLB error ;trap occurred or resul_val not updated
2	L2 insert TLB error. entry not inserted correctly
3	L2 purge TLB error ;entry not purged
4	L2 purge TLB error ;entry purge
5	L2 purge entry TLB error ;entry not purged

SECTION 33:

ERROR NO.	MEANING
1	lwd execution failed in virtual mode
2	ldw not executed for a L2 d-tlb parity error
3	D-TLB MISS not taken (error)
4	incorrect syndrome generated
5	TLB error bit not set
6	DLPMC bit not set

SECTION 34:

LEVEL 1 Instruction TLB Errors:

```
HARD FAULT IN FIRST LEVEL I-TLB LOCKBIT

HARD FAULT IN FIRST LEVEL I-TLB RAM ARRAY

FIRST LEVEL I-TLB ENTRY LOCKED IN BOTH SETS
```

LEVEL 1 Data TLB Errors:

```
HARD FAULT IN FIRST LEVEL D-TLB LOCKBIT

HARD FAULT IN FIRST LEVEL D-TLB RAM ARRAY

FIRST LEVEL D-TLB ENTRY LOCKED IN BOTH SETS
```

SECTION 35:

ERROR NO.	MEANING
1	ITLB instruction not executed.
2	L1 ITLB lock bit not set correctly.
3	L1 ITLB group hit bit not set correctly.
4	DTLB instruction not executed.
5	L1 DTLB lock bit not set correctly.
6	L1 DTLB group hit bit not set correctly.

SECTION 36:

ERROR NO.	MEANING
1	L1 I-TLB rfi error
2	HPMC trap parameter not clear; L1 I-TLB error
3	HPMC PMI field not clear;L1 I-TLB error
4	TOC bit set;L1 I-TLB error
5	instruction address incorrect;L1 I-TLB error
6	PE_BIT not set;L1 I-TLB error
7	test data did not match actual data; L1 I-TLB error
8	HPMC trap parameter not clear; L1 D-TLB error
9	HPMC PMI field not clear;L1 D-TLB error
10	TOC bit set;L1 D-TLB error
11	instruction address incorrect;L1 D-TLB error
12	PE_BIT not set;L1 I-TLB error
13	expected data did not match actual data; L1 I-TLB error

SECTION 37:

ERROR NO.	MEANING
1	L1 I-TLB access rights error
2	L1 D-TLB access rights error
3	L1 I-TLB gateway error
4	L1 I-TLB Protection ID error
5	L1 D-TLB Protection ID error
6	L1 D-TLB write-disable bit error

SECTION 38:

ERROR NO.	MEANING
1	L1 D-TLB B bit error
2	L1 D-TLB D bit error
3	L1 D-TLB T bit error

SECTION 39:

ERROR NO.	MEANING
1	trap error
2	translation error
3	trap error
4	translation error
5	trap error
6	translation error
7	trap error
8	translation error
9	trap error
10	translation error
11	trap error
12	translation error
13	trap error
14	translation error

SECTION 40:

ERROR NO.	MEANING
1	pscq error
2	pscq error
3	virtual translation error
4	virtual translation error
5	ble error
6	ble error
7	privilege level error
8	privilege level error
9	privilege level error
10	privilege level error
11	gateway error
12	privilege level error
13	privilege level error
14	gateway error
15	privilege level error
16	privilege level error
17	gateway error
18	privilege level error

SECTION 041:

ERROR NO	MEANING
1	compare immediate con error
2	compare immediate con error
3	compare immediate con error

SECTION 042:

ERROR NO	MEANING
1	incorrect number of branches

SECTION 043:

ERROR NO	MEANING
1	condition error—never
2	condition error—equal
3	condition error—<
4	condition error—<=
5	condition error—<< unsigned
6	condition error—<<= unsigned
7	condition error—signed overflow
8	condition error—halfword overflow
9	branching error—count is wrong

SECTION 044:

ERROR NO	MEANING
1	offset calculation error

SECTION 045:

ERROR NO	MEANING
1	BV error

SECTION 046:

ERROR NO	MEANING
1	LDIL error
2	LDW error—loaded value
3	LDWM error—loaded value
4	LDWM error—base register
5	LDW error—loaded value (varying displacement)
6	LDWM error—loaded value (varying displacement)
7	ADD gr0,x error

SECTION 047:

ERROR NO	MEANING
2	Error loading pcoq
3	Error loading pcsq
4	Error loading iva register
5	Condition trap error, iva error
6	Error rfi to virtual address
7	Error rfi to virtual address
8	Error trap at virtual address
9	Error pcsq from virtual address
10	Error pcoq from virtual address

SECTION 048:

ERROR NO	MEANING
1	Execute trap error
2	Gateway trap error
3	Gateway privilege change error
4	Gateway error
5	Purge tlb error—acc = 00

SECTION 049:

ERROR NO	MEANING
1	Error LDWAS
2	Error LDWAX
3	PROBER error—priv. level 3
4	PROBERI error—priv. level 3
5	Read (LDW) error—priv. level 3
6	PROBEW error—priv. level 3
7	PROBEWI error—priv. level 3
8	Write (STW) error—priv. level 3
9	PROBER error—priv. level 2
10	PROBERI error—priv. level 2
11	Read (LDW) error—priv. level 2
12	PROBEW error—priv. level 2
13	PROBEWI error—priv. level 2
14	Write (STW) error—priv. level 2
15	PROBER error—priv. level
16	PROBERI error—priv. level
17	Read (LDW) error—priv. level
18	PROBEW error—priv. level
19	PROBEWI error—priv. level
20	Write (STW) error—priv. level
21	PROBER error—priv. level 0
22	PROBERI error—priv. level 0
23	Read (LDW) error—priv. level 0
24	PROBEW error—priv. level 0
25	PROBEWI error—priv. level 0
26	Write (STW) error—priv. level 0
27	Error LPA
28	Error LDW
29	Error LDWM—base register was modified when a trap occurred
30	Error STW
31	Error PDTLB

SECTION 050:

ERROR NO	MEANING
10	BV error

SECTION 051:

ERROR NO	MEANING
1	Conditional nullify error
2	Error extract

SECTION 052:

ERROR NO	MEANING
I	Conditional nullify error
2	Error deposit

SECTION 053:

ERROR NO	MEANING
1	Overflow trap error—addo
2	Conditional nullify error—addo
3	Result error—addo
4	Overflow trap error—addco (carry = 0)
5	Result error—addco (carry = 0)
6	Conditional nullify error—addco (carry = 0)
7	Conditional nullify error—add
8	Overflow trap error—add
9	Conditional nullify error—addc (carry = 0)
10	Result error—addc (carry = 0)
11	Conditional nullify error—addc (carry = 1)
12	Result error—addc (carry = 1)
13	Overflow trap error—addco (carry = 1)
14	Conditional nullify error—addco (carry = 1)
15	Result error—addco (carry = 1)
16	Conditional nullify error—addl
17	Result error—addl

SECTION 054:

ERROR NO	MEANING
1	Condition error
2	Addition error
3	Overflow trap_num error—addio
4	Result error—addio
5	Overflow trap_num error—addito
6	Result error—addito
7	Conditional trap_num error—addit
8	Result error—addit

SECTION 055:

ERROR NO	MEANING
1	condition error—never
2	condition error—= 0
3	condition error—<
4	condition error—<=
8	condition error—odd
9	condition error—always
10	condition error—non zero
11	condition error—>=
12	condition error—>
16	condition error—even
17	logical error

SECTION 056:

ERROR NO	MEANING
1	condition error—equal
2	condition error—<
3	condition error—<=
4	condition error—<<
5	condition error—<<=
6	condition error—signed overflow
7	condition error—odd
8	condition error—never
9	condition error—not equal
10	condition error—>=
11	condition error—>
12	condition error—>>=
13	condition error—>>
14	condition error—no signed overflow
15	condition error—even
16	condition error—always
17	clear error—equal
18	clear error—<
19	clear error—<=
20	clear error—<<
21	clear error—<<=
22	clear error—signed overflow
23	clear error—odd
24	clear error—never
25	clear error—not equal
26	clear error—>=
27	clear error—>
28	clear error—>>=
29	clear error—>>
30	clear error—no signed overflow
31	clear error—even
32	clear error—always

SECTION 057:

ERROR NO	MEANING
1	condition error—never
2	condition error—= 0
3	condition error—<
4	condition error—<=
8	condition error—odd
9	condition error—always
10	condition error—<> 0
11	condition error—>=
12	condition error -
16	condition error—even
17	logical error

SECTION 058:

ERROR NO	MEANING
1	Overflow trap error—subo
2	Conditional nullify error—sub
3	Result error—subo
4	Overflow trap error—subbo (borrow = 1)
5	Result error—subbo (borrow = 1)
6	Conditional nullify error—subbo (borrow = 1)
7	Overflow or conditional trap error—subto
8	Result error—subto
9	Overflow trap error—subt
10	Result error—subt
11	Conditional nullify error—sub
12	Overflow trap error—sub
13	Conditional nullify error—subb (borrow = 0)
14	Result error—subb
15	Conditional nullify error—subb (borrow = 1)
16	Result error—subb (borrow = 1)
17	Overflow trap error—subbo (borrow = 0)
18	Conditional nullify error—subbo (borrow = 0)
19	Result error—subbo (borrow = 0)

SECTION 059:

ERROR NO	MEANING
1	Logical error—uxor
2	Conditional nullify error—uxor
5	Logical error—uxor
6	Conditional nullify error—uxor
9	Logical error—uxor
10	Conditional nullify error—uxor
13	Logical error—uxor
14	Conditional nullify error—uxor
17	Logical error—uxor
18	Conditional nullify error—uxor
21	Logical error—uxor
22	Conditional nullify error—uxor

SECTION 060:

ERROR NO	MEANING
1	condition error—never
2	condition error—= 0
3	condition error—< 0
4	condition error—<= 0
8	condition error—odd
9	condition error—always
10	condition error—non zero
11	condition error—>= 0
12	condition error—> 0
13	condition error—even
14	logical error

SECTION 061:

ERROR NO	MEANING
1	Condition Err:ADDBT,n; ADDBF,n
2	Nullify Err:ADDBT,n; ADDBF,n
3	Result Err:ADDBT,n; ADDBF,n
4	Condition Err:ADDBT; ADDBF
5	Nullify Err:ADDBT; ADDBF
6	Result Err:ADDBT; ADDBF
7	Condition Err:ADDIBT,n; ADDIBF,n
8	Nullify Err:ADDIBT,n; ADDIBF,n
9	Result Err:ADDIBT,n; ADDIBF,
10	Condition Err:ADDIBT; ADDIBF
11	Nullification Err:ADDIBT; ADDIBF
12	Result Err:ADDIBT; ADDIBF

SECTION 062:

ERROR NO	MEANING
1	addil to zero error
2	addil to non-zero value error

SECTION 063:

ERROR NO	MEANING
1	Logical Err:andcm
2	Conditional nullify Err:andcm
3	Logical Err:andcm,=
4	Conditional nullify Err:andcm,=
5	Logical Err:andcm,<
6	Conditional nullify Err:andcm,
7	Logical Err:andcm,<=
8	Conditional nullify Err:andcm,<=
9	Logical Err:andcm,h
10	Conditional nullify Err:andcm,hv
11	Logical Err:andcm,tr
12	Conditional nullify Err:andcm,tr
13	Logical Err:andcm,<>
14	Conditional nullify Err:andcm,<>
15	Logical Err:andcm,>=
16	Conditional nullify Err:andcm,>=
17	Logical Err:andcm,>
18	Conditional nullify Err:andcm,>
19	Logical Err:andcm,nhv
20	Conditional nullify Err:andcm,nhv

SECTION 064:

ERROR NO	MEANING
1	Condition error—DEPI
2	Logical error—DEPI
3	Condition error—ZDEPI
4	Logical error—ZDEPI
5	Condition error—DEP
6	Logical error—DEP
7	Condition error—ZDEP
8	Logical error—ZDEP
9	Condition error—VDEPI
10	Logical error—VDEPI
11	Condition error—ZVDEPI
12	Logical error—ZVDEPI
13	Condition error—ZVDEP
14	Logical error—ZVDEP

SECTION 065:

ERROR NO	MEANING
1	extract error
2	conditional nullify error

SECTION 066:

ERROR NO	MEANING
1	Condition Error—ADDBT,n; ADDBF,n
2	Nullify Error—ADDBT,n; ADDBF,n
3	Result Error—ADDBT,n; ADDBF,n
4	Condition Error—ADDBT; ADDBF
5	Nullify Error—ADDBT; ADDBF
6	Result Error—ADDBT; ADDBF
7	Condition Error—ADDIBT,n; ADDIBF,n
8	Nullify Error—ADDIBT,n; ADDIBF,n
9	Result Error—ADDIBT,n; ADDIBF,n
10	Condition Error—ADDIBT; ADDIBF
11	Nullification Error—ADDIBT; ADDIBF
12	Result Error—ADDIBT; ADDIBF

SECTION 067:

ERROR NO	MEANING
1	Conditional Nullify error—subi
2	Result error—subi
3	Overflow trap error—subio
4	Conditional Nullify error—subio
5	Result error—subio

SECTION 068:

ERROR NO	MEANING
1	Result error—uaddcm
2	Conditional nullify error—uaddcm
3	Result error—uaddcmt
4	Conditional nullify error—uaddcmt

SECTION 069:

ERROR NO	MEANING
1	Illegal instruction trap did not occur

SECTION 070:

ERROR NO	MEANING
1	COMICLR:conditional nullify
2	COMICLR:target reg not cleared

SECTION 071:

ERROR NO	MEANING
1	Error result of dcor
2	Conditional nullify:dcor
3	Conditional nullify:dcor,shc
4	Conditional nullify:dcor,shz
5	Conditional nullify:dcor,sbc
6	Conditional nullify:dcor,sbz
7	Conditional nullify:dcor,tr
8	Conditional nullify:dcor,tr
9	Conditional nullify:dcor,nhc
10	Conditional nullify:dcor,nhz
11	Conditional nullify:dcor,nbc
12	Conditional nullify:dcor,nbz
13	Conditional nullify:dcor,ndc
14	Error result of idcor
15	Conditional nullify:idcor,nv
16	Conditional nullify:idcor,shc
17	Conditional nullify:idcor,shz
18	Conditional nullify:idcor,sbc
19	Conditional nullify:idcor,sbz
20	Conditional nullify:idcor,sdc
21	Conditional nullify:idcor,tr
22	Conditional nullify:idcor,nhc
23	Conditional nullify:idcor,nhz
24	Conditional nullify:idcor,nbc
25	Conditional nullify:idcor,nbz
26	Conditional nullify:idcor,ndc

SECTION 072:

ERROR NO	MEANING
1	condition error on sh?add
2	Result error on sh?add
3	condition error on sh?addo
4	Result error on sh?addo
5	Overflow trap:sh?addo

SECTION 073:

ERROR NO	MEANING
1	VSHD error (position = 0)
2	VSHD conditional nullify error (position = 0)
3	VSHD error
4	VSHD conditional nullify error
5	SHD error
6	SHD conditional nullify error

SECTION 074:

ERROR NO	MEANING
1	ldwax error
2	ldwax modification error
3	ldwax clear error
4	ldwax,m error
5	ldwax,m modification error
6	ldwax,m clear error
7	ldwax,s error
8	ldwax,s modification error
9	ldwax,s clear error
10	ldwax,sm error
11	ldwax,sm modification error
12	ldwax,sm clear error
13	ldwas error
14	ldwas modification error
15	ldwas clear error
16	ldwas,ma error
17	ldwas,ma modification error
18	ldwas,ma clear error
19	ldwas,mb error
20	ldwas,mb modification error
21	ldwas,mb clear error
22	ldwax error
23	ldwax modification error
24	ldwax clear error
25	ldwax,m error
26	ldwax,m modification error
27	ldwax,m clear error
28	ldwax,s error
29	ldwax,s modification error
30	ldwax,s clear error
31	ldwax,sm error
32	ldwax,sm modification error
33	ldwax,sm clear error
34	ldwas error
35	ldwas modification error
36	ldwas clear error
37	ldwas,ma error
38	ldwas,ma modification error
39	ldwas,ma clear error
40	ldwas,mb error
41	ldwas,mb modification error
42	ldwas,mb clear error
43	stwas error
44	stwas modification error
45	stwas clear error
46	stwas,ma error
47	stwas,ma modification error
48	stwas,ma clear error
49	stwas,mb error
50	stwas,mb modification error

SECTION 074 (continued):

51	stwas,mb clear error
52	stwas error
53	stwas modification error
54	stwas clear error
55	stwas,ma error
56	stwas,ma modification error
57	stwas,ma clear error
58	stwas,mb error
59	stwas,mb modification error
60	stwas,mb clear error

SECTION 075:

ERROR NO	MEANING
1	STW shouldn't have stored but did
2	STW shouldn't have trapped but did (X-bit was set)
3	STWM shouldn't have stored but did
4	STWM shouldn't have trapped but did (X-bit was set)
5	STWS shouldn't have stored but did
6	STWS shouldn't have trapped but did (X-bit was set)
7	STB shouldn't have stored but did
8	STB shouldn't have trapped but did (X-bit was set)
9	STBS shouldn't have stored but did
10	STBS shouldn't have trapped but did(X-bit was set)
11	STH shouldn't have stored but did
12	STH shouldn't have trapped but did
13	STHS shouldn't have stored but did
14	STHS shouldn't have trapped but did (X-bit was set)
15	STBYS shouldn't have stored but did
16	STBYS shouldn't have trapped but did (X-bit was set)
17	LDCWS shouldn't have stored but did
18	LDCWS shouldn't have trapped but did (X-bit was set)
19	LDCWX shouldn't have stored but did
20	LDCWX shouldn't have trapped but did (X-bit was set)
21	PDC shouldn't have stored but did
22	PDC shouldn't have trapped but did (X-bit was set)
128	Trap was taken; should not have been
129	Trap was not taken; should have been
130	X-bit was not set
131	X-bit was not cleared

SECTION 076:

ERROR NO	MEANING
1	STW should have trapped but didn't (dirty bit wasn't set)
2	STW shouldn't have stored but did (dirty bit wasn't set)
3	STW shouldn't have trapped but did (dirty bit was set)
4	STW should have stored but didn't (dirty bit was set)
5	STWM should have trapped but didn't (dirty bit wasn't set)
6	STWM shouldn't have stored but did (dirty bit wasn't set)
7	STWM shouldn't have trapped but did (dirty bit was set)
8	STWM should have stored but didn't (dirty bit was set)
9	STWS should have trapped but didn't (dirty bit wasn't set)
10	STWS shouldn't have stored but did (dirty bit wasn't set)
11	STWS shouldn't have trapped but did (dirty bit was set)
12	STWS should have stored but didn't (dirty bit was set)
13	STB should have trapped but didn't (dirty bit wasn't set)
14	STB shouldn't have stored but did (dirty bit wasn't set)
15	STB shouldn't have trapped but did (dirty bit was set)
16	STB should have stored but didn't (dirty bit was set)
17	STBS should have trapped but didn't (dirty bit wasn't set)
18	STBS shouldn't have stored but did (dirty bit wasn't set)
19	STBS shouldn't have trapped but did (dirty bit was set)
20	STBS should have stored but didn't (dirty bit was set)
21	STH should have trapped but didn't (dirty bit wasn't set)
22	STH shouldn't have stored but did (dirty bit wasn't set)
23	STH shouldn't have trapped but did (dirty bit was set)
24	STH should have stored but didn't (dirty bit was set)
25	STHS should have trapped but didn't (dirty bit wasn't set)
26	STHS shouldn't have stored but did (dirty bit wasn't set
27	STHS shouldn't have trapped but did (dirty bit was set)
28	STHS should have stored but didn't (dirty bit was set)
29	STBYS should have trapped but didn't (dirty bit wasn't set)
30	STBYS shouldn't have stored but did (dirty bit wasn't set
31	STBYS shouldn't have trapped but did (dirty bit was set)
32	STBYS should have stored but didn't (dirty bit was set)
33	LDCWS should have trapped but didn't (dirty bit wasn't set)
34	LDCWS shouldn't have stored but did (dirty bit wasn't set)
35	LDCWS shouldn't have trapped but did (dirty bit was set)
36	LDCWS should have stored but didn't (dirty bit was set
37	LDCWX should have trapped but didn't (dirty bit wasn't set)
38	LDCWX shouldn't have stored but did (dirty bit wasn't set)
39	LDCWX shouldn't have trapped but did (dirty bit was set)
40	LDCWX should have stored but didn't (dirty bit was set)

SECTION 077:

ERROR NO	MEANING
1	Error gateway—privilege bits
2	G—Higher priv. trap(3 to 0)
3	G—Higher priv. trap(2 to 0)
4	G—Higher priv. trap(1 to 0)
5	G—Higher priv. trap(3 to)
6	G—Higher priv. trap(2 to)
7	G—Higher priv. trap(3 to 2)
8	BLE—Lower priv. trap(0 to)
9	BLE—Lower priv. trap(0 to 2)
10	BLE—Lower priv. trap(0 to 3)
11	BLE—Lower priv. trap(1 to 2)
12	BLE—Lower priv. trap(1 to 3)
13	BLE—Lower priv. trap(2 to 3)
14	BE—Lower priv. trap(0 to)
15	BE—Lower priv. trap(0 to 2)
16	BE—Lower priv. trap(0 to 3)
17	BE—Lower priv. trap(1 to 2)
18	BE—Lower priv. trap(1 to 3)
19	BE—Lower priv. trap(2 to 3)
20	BV—Lower priv. trap(0 to)
21	BV—Lower priv. trap(0 to 2)
22	BV—Lower priv. trap(0 to 3)
23	BV—Lower priv. trap(1 to 2)
24	BV—Lower priv. trap(1 to 3)
25	BV—Lower priv. trap(2 to 3)
26	privileged register trap not taken
27	privileged register trap not taken

SECTION 078:

ERROR NO	MEANING
1	LDW error—PID1, priv = 0
2	PROBER error—PID1, priv = 0
3	PROBERI error—PID1, priv = 0
4	LDW error—PID1, priv = 1
5	PROBER error—PID1, priv = 1
6	PROBERI error—PID1, priv = 1
7	LDW error—PID1, priv = 2
8	PROBER error—PID1, priv = 2
9	PROBERI error—PID1, priv = 2
10	LDW error—PID1, priv = 3
11	PROBER error—PID1, priv = 3
12	PROBERI error—PID1, priv = 3
13	STW error—PID1, priv = 0
14	PROBEW error—PID1, priv = 0
15	PROBEWI error—PID1, priv = 0
16	STW error—PID1, priv = 1
17	PROBEW error—PID1, priv = 1
18	PROBEWI error—PID1, priv = 1
19	STW error—PID1, priv = 2
20	PROBEW error—PID1, priv = 2
21	PROBEWI error—PID1, priv = 2
22	STW error—PID1, priv = 3
23	PROBEW error—PID1, priv = 3
24	PROBEWI error—PID1, priv = 3
25	PROBER error—PID2, priv = 0
26	PROBERI error—PID2, priv = 0
27	LDW error—PID2, priv = 1
28	PROBER error—PID2, priv = 1
29	PROBERI error—PID2, priv = 1
30	LDW error—PID2, priv = 2
31	PROBER error—PID2, priv = 2
32	PROBERI error—PID2, priv = 2
33	LDW error—PID2, priv = 3
34	PROBER error—PID2, priv = 3
35	PROBERI error—PID2, priv = 3
36	STW error—PID2, priv = 0
37	PROBEW error—PID2, priv = 0
38	PROBEWI error—PID2, priv = 0
39	STW error—PID2, priv = 1
40	PROBEW error—PID2, priv = 1
41	PROBEWI error—PID2, priv = 1
42	STW error—PID2, priv = 2
43	PROBEW error—PID2, priv = 2
44	PROBEWI error—PID2, priv = 2
45	STW error—PID2, priv = 3
46	PROBEW error—PID2, priv = 3
47	PROBEWI error—PID2, priv = 3
48	LDW error—PID3, priv = 0
49	PROBER error—PID3, priv = 0
50	PROBERI error—PID3, priv = 0

SECTION 078 (continued):	
51	LDW error—PID3, priv = I
52	PROBER error—PID3, priv = I
53	PROBERI error—PID3, priv = I
54	LDW error—PID3, priv = 2
55	PROBER error—PID3, priv = 2
56	PROBERI error—PID3, priv = 2
57	LDW error—PID3, priv = 3
58	PROBER error—PID3, priv = 3
59	PROBERI error—PID3, priv = 3
60	STW error—PID3, priv = 0
61	PROBEW error—PID3, priv = 0
62	PROBEWI error—PID3, priv = 0
63	STW error—PID3, priv = I
64	PROBEW error—PID3, priv = 1
65	PROBEWI error—PID3, priv = 1
66	STW error—PID3, priv = 2
67	PROBEW error—PID3, priv = 2
68	PROBEWI error—PID3, priv = 2
69	STW error—PID3, priv = 3
70	PROBEW error—PID3, priv = 3
71	PROBEWI error—PID3, priv = 3
72	LDW error—PID4, priv = 0
73	PROBER error—PID4, priv = 0
74	PROBERI error—PID4, priv = 0
75	LDW error—PID4, priv = 1
76	PROBER error—PID4, priv = 1
77	PROBERI error—PID4, priv = 1
78	LDW error—PID4, priv = 2
79	PROBER error—PID4, priv = 2
80	PROBERI error—PID4, priv = 2
81	LDW error—PID4, priv = 3
82	PROBER error—PID4, priv = 3
83	PROBERI error—PID4, priv = 3
84	STW error—PID4, priv = 0
85	PROBEW error—PID4, priv = 0
86	PROBEWI error—PID4, priv = 0
87	STW error—PID4, priv = 1
88	PROBEW error—PID4, priv = 1
89	PROBEWI error—PID4, priv = 1
90	STW error—PID4, priv = 2
91	PROBEW error—PID4, priv = 2
92	PROBEWI error—PID4, priv = 2
93	STW error—PID4, priv = 3
94	PROBEW error—PID4, priv = 3
95	PROBEWI error—PID4, priv = 3
96	LDW error—NO PID, priv = 0
97	PROBER error—NO PID, priv = 0
98	PROBERI error—NO PID, priv = 0
99	LDW error—NO PID, priv = 1
100	PROBER error—NO PID, priv = 1

SECTION 078 (continued):	
101	PROBERI error—NO PID, priv = 1
102	LDW error—NO PID, priv = 2
103	PROBER error—NO PID, priv = 2
104	PROBERI error—NO PID, priv = 2
105	LDW error—NO PID, priv = 3
106	PROBER error—NO PID, priv = 3
107	PROBERI error—NO PID, priv = 3
108	STW error—NO PID, priv = 0
109	PROBEW error—NO PID, priv = 0
110	PROBEWI error—NO PID, priv = 0
111	STW error—NO PID, priv = 1
112	PROBEW error—NO PID, priv = 1
113	PROBEWI error—NO PID, priv = 1
114	STW error—NO PID, priv = 2
115	PROBEW error—NO PID, priv = 2
116	PROBEWI error—NO PID, priv = 2
117	STW error—NO PID, priv = 3
118	PROBEW error—NO PID, priv = 3
119	PROBEWI error—NO PID, priv = 3
120	PROBEW error—PID1, priv = 0
121	PROBEWI error—PID1, priv = 0
122	PROBEW error—PID1, priv = 1
123	PROBEWI error—PID1, priv = 1
124	PROBEW error—PID1, priv = 2
125	PROBEWI error—PID1, priv = 2
126	PROBEW error—PID1, priv = 3
127	PROBEWI error—PID1, priv = 3
128	PROBEW error—PID2, priv = 0
129	PROBEWI error—PID2, priv = 0
130	PROBEW error—PID2, priv = 1
131	PROBEWI error—PID2, priv = 1
132	PROBEW error—PID2, priv = 2
133	PROBEWI error—PID2, priv = 2
134	PROBEW error—PID2, priv = 3
135	PROBEWI error—PID2, priv = 3
136	PROBEW error—PID3, priv = 0
137	PROBEWI error—PID3, priv = 0
138	PROBEW error—PID3, priv = 1
139	PROBEWI error—PID3, priv = 1
140	PROBEW error—PID3, priv = 2
141	PROBEWI error—PID3, priv = 2
142	PROBEW error—PID3, priv = 3
143	PROBEWI error—PID3, priv = 3
144	PROBEW error—PID4, priv = 0
145	PROBEWI error—PID4, priv = 0
146	PROBEW error—PID4, priv = 1
147	PROBEWI error—PID4, priv = 1
148	PROBEW error—PID4, priv = 2
149	PROBEWI error—PID4, priv = 2
150	PROBEW error—PID4, priv = 3
151	PROBEWI error—PID4, priv = 3

SECTION 079:

ERROR NO	MEANING
1	Execute trap error
2	Gateway trap error
3	Gateway privilege change error new level < Gate level
4	Gateway error new level < old level
5	Purge tlb error—acc = 00 Space = 0, offset = same as physical

SECTION 080:

ERROR NO	MEANING
1	bbf error—bit is one
2	bbt error—bit is one
3	bbf error—bit is zero
4	bbt error—bit is zero
5	bbf error—bit is one
6	bbt error—bit is one
7	bbf error—bit is zero
8	bbt error—bit is zero
9	bbf error—bit is one
10	bbt error—bit is one
11	bbf error—bit is zero
12	bbt error—bit is zero
13	bbf error—bit is one
14	bbt error—bit is one
15	bbf error—bit is zero
16	bbt error—bit is zero
17	bbf error—bit is one
18	bbt error—bit is one
19	bbf error—bit is zero
20	bbt error—bit is zero
21	bbf error—bit is one
22	bbt error—bit is one
23	bbf error—bit is zero
24	bbt error—bit is zero
25	bbf error—bit is one
26	bbt error—bit is one
27	bbf error—bit is zero
28	bbt error—bit is zero
29	bbf error—bit is one
30	bbt error—bit is one
31	bbf error—bit is zero
32	bbt error—bit is zero
33	bbf error—bit is one
34	bbt error—bit is one
35	bbf error—bit is zero
36	bbt error—bit is zero
37	bbf error—bit is one
38	bbt error—bit is one
39	bbf error—bit is zero
40	bbt error—bit is zero
41	bbf error—bit is one
42	bbt error—bit is one
43	bbf error—bit is zero
44	bbt error—bit is zero
45	bbf error—bit is one
46	bbt error—bit is one
47	bbf error—bit is zero
48	bbt error—bit is zero
49	bbf error—bit is one
50	bbt error—bit is one

SECTION 080 (continued):	
51	bbf error—bit is zero
52	bbt error—bit is zero
53	bbf error—bit is one
54	bbt error—bit is one
55	bbf error—bit is zero
56	bbt error—bit is zero
57	bbf error—bit is one
58	bbt error—bit is one
59	bbf error—bit is zero
60	bbt error—bit is zero
61	bbf error—bit is one
62	bbt error—bit is one
63	bbf error—bit is zero
64	bbt error—bit is zero
65	bbf error—bit is one
66	bbt error—bit is one
67	bbf error—bit is zero
68	bbt error—bit is zero
69	bbf error—bit is one
70	bbt error—bit is one
71	bbf error—bit is zero
72	bbt error—bit is zero
73	bbf error—bit is one
74	bbt error—bit is one
75	bbf error—bit is zero
76	bbt error—bit is zero
77	bbf error—bit is one
78	bbt error—bit is one
79	bbf error—bit is zero
80	bbt error—bit is zero
81	bbf error—bit is one
82	bbt error—bit is one
83	bbf error—bit is zero
84	bbt error—bit is zero
85	bbf error—bit is one
86	bbt error—bit is one
87	bbf error—bit is zero
88	bbt error—bit is zero
89	bbf error—bit is one
90	bbt error—bit is one
91	bbf error—bit is zero
92	bbt error—bit is zero
93	bbf error—bit is one
94	bbt error—bit is one
95	bbf error—bit is zero
96	bbt error—bit is zero
97	bbf error—bit is one
98	bbt error—bit is one
99	bbf error—bit is zero
100	bbt error—bit is zero

SECTION 080 (continued):	
101	bbf error—bit is one
102	bbt error—bit is one
103	bbf error—bit is zero
104	bbt error—bit is zero
105	bbf error—bit is one
106	bbt error—bit is one
107	bbf error—bit is zero
108	bbt error—bit is zero
109	bbf error—bit is one
110	bbt error—bit is one
111	bbf error—bit is zero
112	bbt error—bit is zero
113	bbf error—bit is one
114	bbt error—bit is one
115	bbf error—bit is zero
116	bbt error—bit is zero
117	bbf error—bit is one
118	bbt error—bit is one
119	bbf error—bit is zero
120	bbt error—bit is zero
121	bbf error—bit is one
122	bbt error—bit is one
123	bbf error—bit is zero
124	bbt error—bit is zero
125	bbf error—bit is one
126	bbt error—bit is one
127	bbf error—bit is zero
128	bbt error—bit is zero
129	bvbf error—bit is one
130	bvbt error—bit is one
131	bvbf error—bit is zero
132	bvbt error—bit is zero

SECTION 081:

ERROR NO	MEANING
1	Conditional nullify error—nv
2	Conditional nullify error—eq
3	Conditional nullify error—lt
4	Conditional nullify error—le
5	Conditional nullify error—lu
6	Conditional nullify error—leu
7	Conditional nullify error—sv
8	Conditional nullify error—od
9	Conditional nullify error—tr
10	Conditional nullify error—ne
11	Conditional nullify error—ge
12	Conditional nullify error—gt
13	Conditional nullify error—geu
14	Conditional nullify error—gu
15	Conditional nullify error—nsv
16	Conditional nullify error—ev
17	Result error
18	Final V-bit error
19	Carry bits error
20	Result error
21	Final V-bit error
22	Carry bits error
23	Result error
24	Final V-bit error
25	Carry bits error
26	Result error
27	Final V-bit erro
28	Carry bits error

SECTION 082:

ERROR NO	MEANING
1	offset calculation error

SECTION 083:

ERROR NO	MEANING
1	ldb error
2	ldb modification error
5	ldbx error
6	ldbx modification error
7	ldbx,m error
8	ldbx,m modification error
9	ldbx,s error
10	ldbx,s modification error
11	ldbx,sm error
12	ldbx,sm modification error
13	ldbs error
14	ldbs modification error
15	ldbs,ma error
16	ldbs,ma modification error
17	ldbs,mb error
18	ldbs,mb modification error
19	stb error
20	stb modification error
21	stbs error
22	stbs modification error
23	stbs,ma error
24	stbs,ma modification error
25	stbs,mb error
26	stbs,mb modification error
27	struction pdir error
28	data pdir error

SECTION 084:

ERROR NO	MEANING
1	ldw error
2	ldw modification error
3	ldwm error
4	ldwm modification error
5	ldwx error
6	ldwx modification error
7	ldwx,m error
8	ldwx,m modification error
9	ldwx,s error
10	ldwx,s modification error
11	ldwx,sm error
12	ldwx,sm modification error
13	ldws error
14	ldws modification error
15	ldws,ma error
16	ldws,ma modification error
17	ldws,mb error
18	ldws,mb modification error
19	stw error
20	stw modification error
21	stwm error
22	stwm modification error
23	stwx error
24	stwx modification error
25	stws error
26	stws modification error
27	stws,mb error
28	stws,mb modification error
29	stws,ma error
30	stws,ma modification error
31	ldw error using sr
32	ldw error using sr
33	ldw error using sr3
34	ldwx error using sr1
35	ldwx error using sr2
36	ldwx error using sr3
37	Instr PDIR error
38	Data PDIR error
39	Unaligned data ref trap didn't occur
40	Unaligned data ref trap didn't occur

SECTION 085:

ERROR NO	MEANING
1	ldh error
2	ldh modification error
5	ldhx error
6	ldhx modification error
7	ldhx,m error
8	ldhx,m modification error
9	ldhx,s error
10	ldhx,s modification error
11	ldhx,sm error
12	ldhx,sm modification error
13	ldhs error
14	ldhs modification error
15	ldhs,ma error
16	ldhs,ma modification error
17	ldhs,mb error
18	ldhs,mb modification error
19	sth error
20	sth modification error
21	sths error
22	sths modification error
23	sths,ma error
24	sths,ma modification error
25	sths,mb error
26	sths,mb modification error
27	Instr pdir error
28	Data pdir error
29	Unaligned data ref trap didn't occur

SECTION 086:

ERROR NO	MEANING
1	Initial bytes are modified—STBYS,B
2	Stored bytes incorrect—STBYS,B
3	Base register is modified—STBYS,B
4	Initial bytes are modified—STBYS,BM
5	Stored bytes incorrect—STBYS,BM
6	Base reg isn't modified—STBYS,BM
7	Initial bytes are modified—STBYS,E
8	Stored bytes incorrect—STBYS,E
9	Base register is modified—STBYS,E
10	Initial bytes are modified—STBYS,EM
11	Stored bytes incorrect—STBYS,EM
12	Base reg isn't modified—STBYS,EM
13	Instr pdir error
14	Data pdir error

SECTION 087:

ERROR NO	MEANING
1	ldcwx error
2	ldcwx modification error
3	ldcwx clear error
4	ldcwx,m error
5	ldcwx,m modification error
6	ldcwx,m clear error
7	ldcwx,s error
8	ldcwx,s modification error
9	ldcwx,s clear error
10	ldcwx,sm error
11	ldcwx,sm modification error
12	ldcwx,sm clear error
13	ldcws error
14	ldcws modification error
15	ldcws clear error
16	ldcws,ma error
17	ldcws,ma modification error
18	ldcws,ma clear error
19	ldcws,mb error
20	ldcws,mb modification error
21	ldcws,mb clear error
22	Instr pdir error
23	Data pdir error

SECTION 088:

ERROR NO	MEANING
1	wild branch error
2	branch and link error
3	branch and link register error
4	blr error—link value
5	branch vectored error
6	Gateway error
7	Gateway error—nullification
9	branch external error
10	branch and link external error
11	ble error—link value (offset)
12	ble error—link value (sr0)
13	instr pdir error
14	data pdir error

SECTION 089:

ERROR NO	MEANING
1	Clear psw error—ADD
2	Clear psw error—ADDO
3	Clear psw error—SH1ADD
4	Clear psw error—SH2ADD
5	Clear psw error—SH3ADD
6	Clear psw error—SH1ADDO
7	Clear psw error—SH2ADDO
8	Clear psw error—SH3ADDO
9	Clear psw error—ADDC
10	Clear psw error—ADDCO
11	Clear psw error—SUB
12	Clear psw error—SUBO
13	Clear psw error—SUBB
14	Clear psw error—SUBBO
15	Clear psw error—SUBT
16	Clear psw error—SUBTO
17	Clear psw error—SUBI
18	Clear psw error—SUBIO
19	Clear psw error—DS
20	Clear psw error—ADDI
21	Clear psw error—ADDIT
22	Clear psw error—ADDIO
23	Clear psw error—ADDITO
24	Set psw error—ADD
25	Set psw error—ADDO
26	Set psw error—SH1ADD
27	Set psw error—SH2ADD
28	Set psw error—SH3ADD
29	Set psw error—SH1ADDO
30	Set psw error—SH2ADDO
31	Set psw error—SH3ADDO
32	Set psw error—ADDC
33	Set psw error—ADDCO
34	Set psw error—SUB
35	Set psw error—SUB
36	Set psw error—SUBB
37	Set psw error—SUBBO
38	Set psw error—SUBT
39	Set psw error—SUBTO
40	Set psw error—SUBI
41	Set psw error—SUBIO
42	Set psw error—DS
43	Set psw error—ADDI
44	Set psw error—ADDIT
45	Set psw error—ADDIO
46	Set psw error—ADDITO
47	psw[c/b] changed on overflow trap
48	psw[c/b] changed on condition trap
49	psw changed—ADDL
50	psw changed—SH1ADDL

SECTION 089 (continued):

51	psw changed—SH2ADDL
52	psw changed—SH3ADDL
53	psw changed—COMCLR
54	psw changed—OR
55	psw changed—XOR
56	psw changed—AND
57	psw changed—ANDCM
58	psw changed—UXOR
59	psw changed—UADDCM
60	psw changed—UADDCMT
61	psw changed—DCOR
62	psw changed—IDCOR
63	psw changed—COMICLR
64	psw changed—ADDB
65	psw changed—ADDIB
66	psw changed—COMB
67	psw changed—COMIB

SECTION 090:

ERROR NO	MEANING
1	Arith. error 1
2	Arith. error 2
3	Arith. error 3
4	Arith. error 4
5	Arith. error 5
6	Arith. error 6
7	Arith. error 7
8	Arith. error 8
9	Arith. error 9

SECTION 091:

ERROR NO	MEANING
1	AND error
2	OR erro
3	ANDCM error
4	XOR error
5	Nullify error

SECTION 092:

ERROR NO	MEANING
1	SHD error
2	VSHD error
3	Extract error
4	Deposit error
5	Nullify error

SECTION 093:

ERROR NO	MEANING
1	load/store error
2	address modification error
5	LDWAX error
6	address modification error
7	LDCWX error- data
8	address modification error
9	LDWCX error- clear
10	address modification error
11	LDCWS error- data
12	address modification error
13	LDCWS error- clear
14	STBYS error- data
15	PDIRace id error
16	PDIR offset error

SECTION 094:

ERROR NO	MEANING
11	nullify error ADDBT ADDBF COMBT or COMBF
12	branch error ADDBT ADDBF COMBT or COMBF
13	not branch error ADDBT ADDBF COMBT or COMBF
21	nullify error ADDIBT, ADDIBF, COMIBT, and COMIBF
22	branch error ADDIBT, ADDIBF, COMIBT, and COMIBF
23	not branch error ADDIBT, ADDIBF, COMIBT, and COMIBF
31	nullify error BB or BVB under conditions < and >=
32	branch error BB or BVB under conditions < and >=
33	not branch error BB or BVB under conditions < and >=
41	nullify error MOVB or MOVIB
42	branch error MOVB or MOVIB
43	not branch error MOVB or MOVIB
44	return addr. error MOVB or MOVIB
51	nullify error BL or G
52	branch error BL or G
53	not branch error BL or G
54	return addr. error BL or G
61	nullify error BE or BLE
62	branch error BE or BLE
63	not branch error BE or BLE
71	nullify error BLR or BV
72	branch error BLR or BV
73	not branch error BLR or BV
74	return addr. error BLR or BV

SECTION 095:

ERROR NO	MEANING
1	result incorrect, err = n1
2	trap error, err = n2
3	cond. nullify error
4	PCO-front error
5	IIR error
6	PCS-front error
7	ISR error
8	IOR error
9	PCS-back error

SECTION 096:

ERROR NO	MEANING
1	PCOQ error
2	trap error
3	priv. level error
4	data move err
5	space id error
6	addr. mod. error

SECTION 097:

ERROR NO	MEANING
11	trap error
12	PCOQ/IOR error
13	PCSQ/ISR error
21	trap error TLB miss faults
22	PCOQ/IOR error TLB miss faults
23	PCSQ/ISR error TLB miss faults
31	trap error the data memory protection traps caused by LOAD instr
32	PCOQ/IOR error the data memory protection traps caused by LOAD instr
33	PCSQ/ISR error the data memory protection traps caused by LOAD instr
41	trap error the data memory protection traps caused by STORE, LDCWX, LDCWS
42	PCOQ/IOR error data memory protection traps caused by STORE, LDCWX, LDCWS
43	PCSQ/ISR error data memory protection traps caused by STORE, LDCWX, LDCWS
51	trap error data memory break trap with X-bit=0 for STORE, LDCWX, LDCWS
52	PCOQ/IOR error data memory break trap with X-bit=0 for STORE, LDCWX, LDCWS
53	PCSQ/ISR error data memory break trap with X-bit=0 for STORE, LDCWX, LDCWS
61	trap error TLB dirty fault caused by various STORE, LDCWX, LDCWS
62	PCOQ/IOR error TLB dirty fault caused by various STORE, LDCWX, LDCWS
63	PCSQ/ISR error TLB dirty fault caused by various STORE, LDCWX, LDCWS
71	trap error TLB miss fault caused by various STORE, LDCWX,LDCWS, and LOAD
72	PCOQ/IOR error TLB miss fault in STORE, LDCWX,LDCWS, and LOAD
73	PCSQ/ISR error TLB miss fault in STORE, LDCWX,LDCWS, and LOAD
81	trap error non-access data TLB miss fault for PROBE
82	PCOQ/IOR error non-access data TLB miss fault for PROBE
83	PCSQ/ISR error non-access data TLB miss fault for PROBE
84	PROBE access error non-access data TLB miss fault for PROBE
91	trap error non-access data TLB miss fault for LPA
92	PCOQ/IOR error non-access data TLB miss fault for LPA
93	PCSQ/ISR error non-access data TLB miss fault for LPA
94	PROBE access error non-access data TLB miss fault for LPA
95	LPA phy addr error
96	LPA addr mod error

SECTION 098:

ERROR NO	MEANING
1	correct result in UXOR, UADDCM, DCOR, IDCOR, or DS
2	nullify error UXOR, UADDCM, DCOR, IDCOR, or DS
3	PSW[V] error UXOR, UADDCM, DCOR, IDCOR, or DS

Note

For the floating point tests, any error numbers that are higher in value than those listed, are extended tests. Extended tests print out information concerning the error in the following manner:

```
Instr: 0000000000000000  Rounding Mode: to 000000000000
Operand 1: 0x00000000 00000000  Operand 2: 0x00000000 00000000
Hardware Result: 0x00000000 00000000  Status: 0x00000000
Software Result: 0x00000000 00000000  Status: 0x00000000
```

The zeroes, of course, will be filled in with the appropriate information Sections 120 and 121 have a special error message format that differs slightly from the previous format. The flops being used, as well as the number of nops between them, will be printed out.

SECTION 099:

ERROR NO	MEANING
1	FCPY of freg0 (as source) failed with an exception trap (single)
2	FCPY of freg to freg0 (as destination) failed to exception trap immeadiate
3	FCPY non-zero freg to non-zero freg failed with an exception trap (single)
4	FCPY of freg0 (as source) failed with an exception trap (double)
5	FCPY of freg to freg0 (as destination) failed to exception trap immeadiate
6	FCPY non-zero freg to non-zero freg failed with an exception trap (double)
7	results from previous operations are incorrect

SECTION 100:

ERROR NO	MEANING
1	FABS of freg0 (as source) failed with an exception trap (double)
2	FABS of freg1 (as source) failed with an exception trap (double)
3	FABS of freg2 (as source) failed with an exception trap (double)
4	FABS of freg3 (as source) failed with an exception trap (double)
5	FABS of freg4 (as source) failed with an exception trap (double)
6	FABS of freg5 (as source) failed with an exception trap (double)
7	FABS of freg6 (as source) failed with an exception trap (double)
8	FABS of freg7 (as source) failed with an exception trap (double)
9	FABS of freg8 (as source) failed with an exception trap (double)
10	FABS of freg9 (as source) failed with an exception trap (double)
11	FABS of freg10 (as source) failed with an exception trap (double)
12	FABS of freg11 (as source) failed with an exception trap (double)
13	FABS of freg12 (as source) failed with an exception trap (double)
14	FABS of freg13 (as source) failed with an exception trap (double)
15	FABS of freg14 (as source) failed with an exception trap (double)
16	FABS of freg15 (as source) failed with an exception trap (double)
17	FABS of neg number (as source) to freg 0 failed to take immeadiate exception
18	FABS of neg number (as source) to freg 1 failed to take immeadiate exception
19	FABS of neg number (as source) to freg 2 failed to take immeadiate exception
20	FABS of neg number (as source) to freg 3 failed to take immeadiate exception
21	FABS of neg number (as source) to freg4 failed with an exception trap
22	FABS of neg number (as source) to freg5 failed with an exception trap
23	FABS of neg number (as source) to freg6 failed with an exception trap
24	FABS of neg number (as source) to freg7 failed with an exception trap
25	FABS of neg number (as source) to freg8 failed with an exception trap
26	FABS of neg number (as source) to freg9 failed with an exception trap
27	FABS of neg number (as source) to freg10 failed with an exception trap
28	FABS of neg number (as source) to freg11 failed with an exception trap
29	FABS of neg number (as source) to freg12 failed with an exception trap
30	FABS of neg number (as source) to freg13 failed with an exception trap
31	FABS of neg number (as source) to freg14 failed with an exception trap
32	FABS of neg number (as source) to freg15 failed with an exception trap
33	Failed to take exception trap on FABS with quad format
34	Failed to take exception trap on FABS with reserve freg (as source) (single)
35	Failed to take exception trap on FABS with reserve freg (as source) (double)
36	FABS of neg number (as source) to single format failed with exception trap
37	FABS of pos number (as source) to single format failed with exception trap
38	FABS of neg number (as source) to reseve reg failed to take exception trap
39	FABS of freg0 (as source) to single format failed with exception trap
40	results of previous operations are incorrect

SECTION 101:

ERROR NO	MEANING
1	FCMP of reserved freg failed to take exception trap
2	FCMP of freg with itself fails to equal
4	FCMP with format code=2 fails to take exception trap
5	FCMP of freg with reserved freg failed to take exception trap
6	a single class2 flop with subop=3 failed to take exception trap
7	a single class2 flop with subop=2 failed to take exception trap
8	a single class2 flop with subop=4 failed to take exception trap
9	FCMP of two different fregs with different values failed to differ
10	FCMP of two different fregs with different values fails with exception trap
11	FCMP of two different fregs with equivalent value fails with exception trap
12	FCMP of two different fregs with equivalent value failed to equal
13	FCMP of two different fregs with equal value fails with exception trap (dbl)
14	FCMP of two different fregs with equal value fails with exception trap (dbl)
15	class2 flop,dbl subop=2 fails to trap immeadiately with exception trap
16	subsequent store failed with exception trap
17	FCMP of two fregs with diff values compared as equal
18	exception trap occurred on previous operation
19	subsequent store failed with exception trap
20	class2 flop dbl subop=1 n=0 failed with exception trap
21	subsequent store failed with exception trap
22	FCMP with reserve freg fails to trap with exception trap
23	Failed to raise exception trap on unimplemeted frem instruction
24	exception trap occurred for previous test, but error not marked unimplemeted
25	same as error 23 above, but for double precision
26	same as error 24 above, but for double precision
27	results from previous tests are not correct

SECTION 102:

ERROR NO	MEANING
1	failed to FADD freg 0 via an exception trap
2-10	FADD failed with exception trap
11-17	FADD fails to take unimplemeted exception trap due to underflow
18	FADD takes exception trap on overflow with overflows dissabled
19	FADD fails with exception trap on inexact
20	FADD fails to take exception trap on invalid condition
21	FADD fails on underflow condition; overflow enabled
22	FADD fails on overflow condition
23	FADD fails on inexact condition
24	FADD fails on invalid condition
25	FADD fails on underflow condition; inexact enabled
26	FADD fails on overflow condition
27	FADD fails on inexact condition
28	FADD fails on invalid condition
29	FADD fails on underflow condition
30	FADD fails on overflow condition
31	FADD fails on inexact condition
32	FADD fails on invalid condition
33	FADD fails on inexact and invalid condition
34	FADD fails on overflow and inexact condition
35	FADD fails on invalid condition
36	FADD fails with exception when one of the fregs is reserved
37	results from previous tests are not correct

SECTION 103:

ERROR NO	MEANING
1	FADD failed with freg 0 as one argumet, trapped with an exception trap (dbl)
2	FADD failed with freg 0 as one argumet, trapped with an exception trap (dbl)
3-11	FADD failed with exception trap
12	FADD fails on underflow condition, all traps enabled (dbl)
13	FADD fails on overflow condition (dbl)
14	FADD fails on inexact condition (dbl)
15	FADD fails on invalid condition (dbl)
16	FADD fails on underflow condition (dbl)
17	FADD fails on overflow condition (dbl)
18	FADD fails on inexact condition (dbl)
19	FADD fails on invalid condition (dbl)
20	FADD fails on underflow and inexact condition, inexact and overflow enabled
21	FADD fails on overflow and inexact condition (dbl)
22	FADD fails on invalid condition (dbl)
23-29	FADD failed with an exception trap
30	results from previous test are incorrect

SECTION 104:

ERROR NO	MEANING
1-2	FSUB failed with an exception trap (sgl)
3	FSUB with one argument freg 0 failed with an exception trap (sgl)
4	FSUB failed with an exception trap (dbl)
5	results from previous tests failed

SECTION 105:

ERROR NO	MEANING
1	FCMP cc = 0, false? failed (dbl)
2	FCMP cc = 0, false? failed with exception trap
3	FCMP cc = 1, false failed (dbl)
4	FCMP cc = 1, false failed with exception trap
5	FCMP cc =2, ? failed (dbl)
6	FCMP cc =2, ? failed with exception trap
7	FCMP cc = 3, !<=> failed (dbl)
8	FCMP cc = 4, = failed (dbl)
9	FCMP cc = 4, = failed with exception trap
10	FCMP cc = 5, =T failed (dbl)
11	FCMP cc = 5, =T failed with exception trap
12	FCMP cc = 6, ?= failed (dbl)
13	FCMP cc = 6, ?= failed with exception trap
14	FCMP cc = 7, !<> failed (dbl)
15	FCMP cc = 7, !<> failed with exception trap
16	FCMP cc = 8, !?>= failed (dbl)
17	FCMP cc = 8, !?>= failed with exception trap
18	FCMP cc = 9, < failed (dbl)
19	FCMP cc = 9, < failed with exception trap
20	FCMP cc = 10, ?< failed (dbl)
21	FCMP cc = 10, ?< failed with exception trap
22	FCMP cc = 11, !>= failed (dbl)
23	FCMP cc = 11, !>= failed with exception trap
24	FCMP cc = 12, !?> failed (dbl)
25	FCMP cc = 12, !?> failed with exception trap
26	FCMP cc = 13, <= failed (dbl)
27	FCMP cc = 13, <= failed with exception trap
28	FCMP cc = 14, ?<= failed (dbl)
29	FCMP cc = 14, ?<= failed with exception trap

30 FCMP cc = 15, !> failed (dbl)
31 FCMP cc = 15, !> failed with exception trap
32 FCMP cc = 16, !?<= failed (dbl)
33 FCMP cc = 16, !?<= failed with exception trap
34 FLDDS follwing previous flop failed
35 FCMP cc = 17, > failed (dbl)
36 FCMP cc = 17, > failed with exception trap
37 FLDDS follwing previous flop failed
38 FCMP cc = 18, ?> failed to trap with an exception trap
39 FSTDS of status freg following previous flop trapped with exception
40 FCMP cc = 19, !<= failed to trap with an exception trap
41 FSTDS of status freg following previous flop trapped with exception
42 FCMP cc = 20, !?< failed (dbl)
43 FCMP cc = 20, !?< failed with exception trap
44 FLDDS follwing previous flop failed
45 FCMP cc = 21, >= failed (dbl)
46 FCMP cc = 21, >= failed with exception trap
47 FLDDS follwing previous flop failed
48 FCMP cc = 22, ?>= failed (dbl)
49 FCMP cc = 22, ?>= failed with exception trap
50 FLDDS follwing previous flop failed
51 FCMP cc = 23, !< failed (dbl)
52 FCMP cc = 23, !< failed with exception trap
53 FLDDS follwing previous flop failed
54 FCMP cc = 24, !?= failed (dbl)
55 FCMP cc = 24, !?= failed with exception trap
56 FLDDS follwing previous flop failed
57 FCMP cc = 25, <> failed (dbl)
58 FCMP cc = 25, <> failed with exception trap
59 FLDDS follwing previous flop failed
60 FCMP cc = 26, != failed (dbl)
61 FCMP cc = 26, != failed with exception trap
62 FLDDS follwing previous flop failed
63 FCMP cc = 27, !=T failed (dbl)
64 FCMP cc = 27, !=T failed with exception trap
65 FLDDS follwing previous flop failed
66 FCMP cc = 28, !? failed (dbl)
67 FCMP cc = 28, !? failed with exception trap
68 FLDDS follwing previous flop failed
69 FCMP cc = 29, <=> failed (dbl)
70 FCMP cc = 29, <=> failed with exception trap
71 FLDDS follwing previous flop failed
72 FCMP cc = 30, true? failed (dbl)
73 FCMP cc = 30, true? failed with exception trap
74 FLDDS follwing previous flop failed
75 FCMP cc = 31, true failed (dbl)
76 FCMP cc = 31, true failed with exception trap
77 results from previuos operations are incorrect

SECTION 106:

ERROR NO	MEANING
1	FCMP cc = 0, false? failed (sgl)
2	FCMP cc = 0, false? failed with exception trap
3	FLDWS follwing previous flop failed
4	FCMP cc = 1, false failed (sgl)
5	FCMP cc = 1, false failed with exception trap
6	FLDWS follwing previous flop failed
7	FCMP cc = 2, ? failed (sgl)
8	FCMP cc = 2, ? failed with exception trap
9	FLDWS follwing previous flop failed
10	FCMP cc = 3, !<=> failed (sgl)
11	FCMP cc = 3, !<=> failed with exception trap
12	FLDWS follwing previous flop failed
13	FCMP cc = 4, = failed (sgl)
14	FCMP cc = 4, = failed with exception trap
15	FLDWS follwing previous flop failed
16	FCMP cc = 5, =T failed (sgl)
17	FCMP cc = 5, =T failed with exception trap
18	FLDWS follwing previous flop failed
19	FCMP cc = 6, ?= failed (sgl)
20	FCMP cc = 6, ?= failed with exception trap
21	FLDWS follwing previous flop failed
22	FCMP cc = 7, !<> failed (sgl)
23	FCMP cc = 7, !<> failed with exception trap
24	FLDWS follwing previous flop failed
25	FCMP cc = 8, !?>= failed (sgl)
26	FCMP cc = 8, !?>= failed with exception trap
27	FLDWS follwing previous flop failed
28	FCMP cc = 9, < failed (sgl)
29	FCMP cc = 9, < failed with exception trap
30	FLDWS follwing previous flop failed
31	FCMP cc = 10, ?< failed (sgl)
32	FCMP cc = 10, ?< failed with exception trap
33	FLDWS follwing previous flop failed
34	FCMP cc = 11, !>= failed (sgl)
35	FCMP cc = 11, !>= failed with exception trap
36	FLDWS follwing previous flop failed
37	FCMP cc = 12, !?> failed to trap with an exception
38	FSTDS of freg 0 failed with an exception trap
39	FCMP cc = 13, <= failed (sgl)
40	FCMP cc = 13, <= failed with exception trap
41	FLDWS follwing previous flop failed
42	FCMP cc = 14, ?<= failed (sgl)
43	FCMP cc = 14, ?<= failed with exception trap
44	FLDWS follwing previous flop failed
45	FCMP cc = 15, !> failed (sgl)
46	FCMP cc = 15, !> failed with exception trap
47	FCMP cc = 16, !?<= failed (sgl)
48	FCMP cc = 16, !?<= failed with exception trap
49	FLDWS follwing previous flop failed
50	FCMP cc = 17, > failed (sgl)

51	FCMP cc = 17, > failed with exception trap
52	FLDWS follwing previous flop failed
53	FCMP cc = 18, ?> failed (sgl)
54	FCMP cc = 18, ?> failed with exception trap
55	FLDWS follwing previous flop failed
56	FCMP cc = 19, !<= failed (sgl)
57	FCMP cc = 19, !<= failed with exception trap
58	FLDWS follwing previous flop failed
59	FCMP cc = 20, !?< failed (sgl)
60	FCMP cc = 20, !?< failed with exception trap
61	FLDWS follwing previous flop failed
62	FCMP cc = 21, >= failed (sgl)
63	FCMP cc = 21, >= failed with exception trap
64	FLDWS follwing previous flop failed
65	FCMP cc = 22, ?>= failed (sgl)
66	FCMP cc = 22, ?>= failed with exception trap
67	FLDWS follwing previous flop failed
68	FCMP cc = 23, !< failed (sgl)
69	FCMP cc = 23, !< failed with exception trap
70	FLDWS follwing previous flop failed
71	FCMP cc = 24, !?= failed (sgl)
72	FCMP cc = 24, !?= failed with exception trap
73	FLDWS follwing previous flop failed
74	FCMP cc = 25, <> failed (sgl)
75	FCMP cc = 25, <> failed with exception trap
76	FLDWS follwing previous flop failed
77	FCMP cc = 26, != failed (sgl)
78	FCMP cc = 26, != failed with exception trap
79	FLDWS follwing previous flop failed
80	FCMP cc = 27, !=T failed (sgl)
81	FCMP cc = 27, !=T failed with exception trap
82	FLDWS follwing previous flop failed
83	FCMP cc = 28, !? failed (sgl) to trap with an exception trap
84	store of freg 0 failed with an exception trap
85	FCMP cc = 29, <=> failed (sgl)
86	FCMP cc = 29, <=> failed with exception trap
87	FLDWS follwing previous flop failed
88	FCMP cc = 30, true? failed (sgl)
89	FCMP cc = 30, true? failed with exception trap
90	FLDWS follwing previous flop failed
91	FCMP cc = 31, true failed (sgl)
92	FCMP cc = 31, true failed with exception trap
93	results from previous operations are incorrect

SECTION 107:

ERROR NO	MEANING
1	FCMP cc = 0, false? failed
2	FCMP cc = 0, false? failed with exception trap
3	FLDWS follwing previous flop failed
4	FCMP cc = 1, false failed
5	FCMP cc = 1, false failed with exception trap
6	FLDWS follwing previous flop failed
7	FCMP cc = 2, ? failed
8	FCMP cc = 2, ? failed with exception trap
9	FLDWS follwing previous flop failed
10	FCMP cc = 3, !<=> failed
11	FCMP cc = 3, !<=> failed with exception trap
12	FLDWS follwing previous flop failed
13	FCMP cc = 4, = failed
14	FCMP cc = 4, = failed with exception trap
15	FLDWS follwing previous flop failed
16	FCMP cc = 5, =T failed
17	FCMP cc = 5, =T failed with exception trap
18	FLDWS follwing previous flop failed
19	FCMP cc = 6, ?= failed
20	FCMP cc = 6, ?= failed with exception trap
21	FLDWS follwing previous flop failed
22	FCMP cc = 7, !<> failed
23	FCMP cc = 7, !<> failed with exception trap
24	FLDWS follwing previous flop failed
25	FCMP cc = 8, !?>= failed
26	FCMP cc = 8, !?>= failed with exception trap
27	FLDWS follwing previous flop failed
28	FCMP cc = 9, < failed
29	FCMP cc = 9, < failed with exception trap
30	FLDWS follwing previous flop failed
31	FCMP cc = 10, ?< failed
32	FCMP cc = 10, ?< failed with exception trap
33	FLDWS follwing previous flop failed
34	FCMP cc = 11, !>= failed to trap with an exception
35	FSTDS of freg 0 failed with an exception trap
36	FCMP cc = 12, !?> failed to trap with an exception
37	FSTDS of freg 0 failed with an exception trap
38	FCMP cc = 13, <= failed
39	FCMP cc = 13, <= failed with exception trap
40	FLDWS follwing previous flop failed
41	FCMP cc = 14, ?<= failed
42	FCMP cc = 14, ?<= failed with exception trap
43	FLDWS follwing previous flop failed
44	FCMP cc = 15, !> failed
45	FCMP cc = 15, !> failed with exception trap
46	FCMP cc = 16, !?<= failed
47	FCMP cc = 16, !?<= failed with exception trap
48	FLDWS follwing previous flop failed
49	FCMP cc = 17, > failed
50	FCMP cc = 17, > failed with exception trap

51	FLDWS follwing previous flop failed
52	FCMP cc = 18, ?> failed
53	FCMP cc = 18, ?> failed with exception trap
54	FLDWS follwing previous flop failed
55	FCMP cc = 19, !<= failed
56	FCMP cc = 19, !<= failed with exception trap
57	FLDWS follwing previous flop failed
58	FCMP cc = 20, !?< failed
59	FCMP cc = 20, !?< failed with exception trap
60	FLDWS follwing previous flop failed
61	FCMP cc = 21, >= failed
62	FCMP cc = 21, >= failed with exception trap
63	FLDWS follwing previous flop failed
64	FCMP cc = 22, ?>= failed
65	FCMP cc = 22, ?>= failed with exception trap
66	FLDWS follwing previous flop failed
67	FCMP cc = 23, !< failed
68	FCMP cc = 23, !< failed with exception trap
69	FLDWS follwing previous flop failed
70	FCMP cc = 24, !?= failed
71	FCMP cc = 24, !?= failed with exception trap
72	FLDWS follwing previous flop failed
73	FCMP cc = 25, <> failed
74	FCMP cc = 25, <> failed with exception trap
75	FLDWS follwing previous flop failed
76	FCMP cc = 26, != failed
77	FCMP cc = 26, != failed with exception trap
78	FLDWS follwing previous flop failed
79	FCMP cc = 27, !=T failed to take an exception trap
80	store freg 0 failed with exception trap
81	FCMP cc = 28, !? failed to trap with an exception trap
82	store of freg 0 failed with an exception trap
83	FCMP cc = 29, <=> failed
84	FCMP cc = 29, <=> failed with exception trap
85	FLDWS follwing previous flop failed
86	FCMP cc = 30, true? failed
87	FCMP cc = 30, true? failed with exception trap
88	FLDWS follwing previous flop failed
89	FCMP cc = 31, true failed
90	FCMP cc = 31, true failed with exception trap
91	results from previous operations are incorrect

SECTION 108:

ERROR NO	MEANING
1	FCMP cc = 0, false? failed to take exception trap
2	store of freg 0 failed with an exception trap
3	FCMP cc = 1, false failed to take exception trap
4	store of freg 0 failed with an exception trap
5	FCMP cc = 2, ? failed to take exception trap
6	store of freg 0 failed with an exception trap
7	FCMP cc = 3, !<=> failed to take exception trap
8	store of freg 0 failed with an exception trap
9	FCMP cc = 4, = failed to take exception trap
10	store of freg 0 failed with an exception trap
11	FCMP cc = 5, =T failed to take exception trap
12	store of freg 0 failed with an exception trap
13	FCMP cc = 6, ?= failed to take exception trap
14	store of freg 0 failed with an exception trap
15	FCMP cc = 7, !<> failed to take exception trap
16	store of freg 0 failed with an exception trap
17	FCMP cc = 8, !?>= failed to take exception trap
18	store of freg 0 failed with an exception trap
19	FCMP cc = 9, < failed to take exception trap
20	store of freg 0 failed with an exception trap
21	FCMP cc = 10, ?< failed to take exception trap
22	store of freg 0 failed with an exception trap
23	FCMP cc = 11, !>= failed to take exception trap
24	store of freg 0 failed with an exception trap
25	FCMP cc = 12, !?> failed to take exception trap
26	store of freg 0 failed with an exception trap
27	FCMP cc = 13, <= failed to take exception trap
28	store of freg 0 failed with an exception trap
29	FCMP cc = 14, ?<= failed to take exception trap
30	store of freg 0 failed with an exception trap
31	FCMP cc = 15, !> failed to take exception trap
32	store of freg 0 failed with an exception trap
33	FCMP cc = 16, !?<= failed to take exception trap
34	store of freg 0 failed with an exception trap
35	FCMP cc = 17, > failed to take exception trap
36	store of freg 0 failed with an exception trap
37	FCMP cc = 18, ?> failed to take exception trap
38	store of freg 0 failed with an exception trap
39	FCMP cc = 19, !<= failed to take exception trap
40	store of freg 0 failed with an exception trap
41	FCMP cc = 20, !?< failed to take exception trap
42	store of freg 0 failed with an exception trap
43	FCMP cc = 21, >= failed to take exception trap
44	store of freg 0 failed with an exception trap
45	FCMP cc = 22, ?>= failed to take exception trap
46	store of freg 0 failed with an exception trap
47	FCMP cc = 23, !< failed to take exception trap
48	store of freg 0 failed with an exception trap
49	FCMP cc = 24, !?= failed to take exception trap
50	store of freg 0 failed with an exception trap

51	FCMP cc = 25, <> failed to take exception trap
52	store of freg 0 failed with an exception trap
53	FCMP cc = 26, != failed to take exception trap
54	store of freg 0 failed with an exception trap
55	FCMP cc = 27, !=T failed to take exception trap
56	store of freg 0 failed with an exception trap
57	FCMP cc = 28, !? failed to take exception trap
58	store of freg 0 failed with an exception trap
59	FCMP cc = 29, <=> failed to take exception trap
60	store of freg 0 failed with an exception trap
61	FCMP cc = 30, true? failed to take exception trap
62	store of freg 0 failed with an exception trap
63	FCMP cc = 31, true failed to take exception trap
64	store of freg 0 failed with an exception trap
65	results of previous operations are incorrect

SECTION 109:

ERROR NO	MEANING
1	failed to take exception trap on FCNVFF,sgl,dbl with reserve freg as source
2	FCNVFF,sgl,dbl failed with freg 0 as destination
3-4	FCNVFF,sgl,dbl failed with an exception trap
5	results of previous operations are incorrect

SECTION 110:

ERROR NO	MEANING
1	failed to take exception trap on FCNVFF,dbl,sgl with reserve freg as source
2	FCNVFF,dbl,sgl failed with freg 0 as destination
3-7	FCNVFF,dbl,sgl failed with an exception trap
8-9	FCNVFF,dbl,sgl failed with an exception trap with all traps enabled
10	FCNVFF,dbl,quad failed to take an exception trap
11	FCNVFF,sgl,dbl failed with an exception trap
12	FCNVFF,sgl,quad failed to take an exception trap
13	FCNVFF,sgl,dbl failed with an exception trap
14	exception trap occurred unexpectedly on FCNVFF with bad format (convert sgl)
15	FCMP,dbl,!>= failed with exception
16	FCNVFF,dbl,sgl with freg 0 as source failed with exception trap
17	results from above operations are incorrect

SECTION 111:

ERROR NO	MEANING
	This section has extended test error messages

SECTION 112:

ERROR NO	MEANING
1-2	FCNVFX,dbl,dbl failed with exception trap
3	FCNVFX,dbl,dbl failed to take exception trap
4	FCNVFX,dbl,dbl failed with exception trap
5	FCNVFX,dbl,dbl failed to take exception trap
6	FCNVFX,dbl,dbl with source x ($\|x\| < 1$) failed with exception trap
7	FCNVFX,dbl,dbl with source x ($\|x\| < 1$) failed to take exception trap
8	results from previous operations are incorrect

SECTION 113:

ERROR NO	MEANING
1	FCNVFXT,sgl,sgl failed with an exception trap
2	results from previous operations are incorrect

SECTION 114:

ERROR NO	MEANING
	all tests are extended test

SECTION 115:

ERROR NO	MEANING
1	FCNVXF,sgl,sgl failed to take an exception trap with a reserve freg as source
2	FCNVXF,sgl,sgl failed to take an exception trap with a reserve freg as dest.
3	store to freg 0 did not reset T bit after reserve format FCNVXF
4	results from previous operations are incorrect

SECTION 116:

ERROR NO	MEANING
	all tests are extended test

SECTION 117:

ERROR NO	MEANING
1	FMPY,sgl failed with an exception trap
2	FMPY,sgl failed to take an exception trap
3	FMPY,dbl failed with an exception trap
4	FMPY,dbl failed to take an exception trap
5	results from previous operations are incorrect

SECTION 118:

ERROR NO	MEANING
1-3	FDIV,sgl failed with an exception trap
4	FDIV,dbl failed with an exception trap
5-10	FDIV,dbl failed with an exception trap with freg zero as a source
11	FDIV,dbl failed with an exception trap
12-14	FDIV,dbl failed with an exception trap with freg zero as a source
15	FDIV,dbl failed to take an exception trap
16	results from previous operations are incorrect

SECTION 119:

ERROR NO	MEANING
	all tests are extended test

SECTION 120:

ERROR NO	MEANING
	all tests are extended test

SECTION 121:

ERROR NO	MEANING
	all tests are extended test

SECTION 122:

ERROR NO	MEANING
1	FSQRT,dbl failed with an exception trap
2	FSQRT,dbl with freg0 as source, failed with an exception trap
3	FSQRT,dbl failed with an exception trap
4	back to back FSQRT,dbl failed with an exception trap
5	back to back FSQRT,dbl failed to take an exception trap
6	results from previous operations are incorrect

Contents

10

Multi-Processor Diagnostic (MPROC)

Introduction

The Multiprocessor Diagnostic (MPROC) is designed to test the PCX-SMB chip set, to verify cache and TLB coherency, as well as arbitration and shared memory accesses; it is designed to conform to a Master/Slave paired structure. It does not test N processors simultaneously.

Defects and Enhancements

Submit defect reports and enhancement requests concerning this diagnostic through the STARS database, referencing product number 30343-10006 (MPROC).

Minimum Configuration

The following system components are necessary to run MPROC:

- A disk or tape
- Two HP 980/870 processor boards set up for MP
- Known good working PDC

Using MPROC

To invoke MPROC, type the following at the prompt:

```
ISL>mproc
```

Once MPROC has been invoked, type an appropriate command at the prompt; for example:

```
MPROC>resume
```

or

```
MPROC>reset
```

or

```
MPROC>default
```

Which command you enter will depend on how you wish the diagnostic to run. See the following section for more details.

The User Interface

The following changes have been made to the standard SUIS for MPROC:

- ISL's RUN command has been renamed to RESET (since it is performing a soft-boot); there is no conflict. This command will also make the *MONARCH* processor the *MASTER*, and places the other processors in the *IDLE* state.
- ISL's RESET command has been renamed to DEFAULT. This command makes the *MONARCH* processor the *MASTER*, and places the other processors in the *IDLE* state.

Note

These changes result in one major difference between the MPROC command set and the standard SUIS command set: MPROC does not have a RUN command, in the SUIS sense of it.

Processor Command Summary

Following is a list of valid processor commands:

changeio	Change paths to boot devices, console, or printer
default	Re-initialize runtime options to default values
eepr	Enable error message printing
eeps	Enable a pause after an error is detected
eipr	Enable isolation message printing
eips	Enable a pause after an isolation message is printed
enpr	Enable non-error message printing
enps	Enable a pause after a non-error message is printed
erro	Shorthand for **eepr;sipr;snpr**
errp	Shorthand for **epps;sips;snps**

exit	Put IDLE processors into PDC_RENDEZVOUS and return the *MONARCH* processor to ISL
freg	Display contents of the floating point registers for each of the processors at the end of last test
hard	Force all messages to a hardcopy device
help	Print information on operating the diagnostic
idle n	Sets a single processor currently in the slave state to the idle state
info	Print information about diagnostic (e.g., what sections test what)
listio	List the I/O paths to boot devices, console, and printer
loop<n>	Set number of times to loop through diagnostic
mast n	Swaps a single processor currently in idle state to the master state
pstatus	Display processor chip revision number, and cache line lock for each of the processors
registers	Display general and control registers at end of last test for each of the processors
reset	Re-initialize runtime options to default values
resume	Restart diagnostic after a pause
sect R	Set numbers of test sections to be executed by master (and slave, if set) where R is a range
sepr	Suppress error message printing
seps	Suppress a pause after an error is detected
sipr	Suppress isolation message printing
sips	Suppress a pause after an isolation message is printed
slav n	Sets a single processor currently in idle state to the slave state
snpr	Suppress non-error message printing
snps	Suppress a pause after a non-error message is printed
state	Display the current values of the runtime options
stop	Stop ISL from continuing the auto boot sequence

General Troubleshooting Strategy

MPROC is strictly a multi-processor diagnostic. It does not encompass the tests that the uni-processor diagnostics (such as proc_diag or the A1100AP series) do. Testing should be performed in a reverse-onion skin approach, eliminating the less complicated errors first. Therefore, the normal flow of test execution is:

1. Self Test
2. Uni-Processor Standalone Diagnostics (such as proc_diag, memdiag)
3. Multi-Processor Standalone Diagnostics (MPROC)
4. O.S. Based Tests

It is expected that before a user runs **MPROC** they have already determined, through the use of the Uni-Processor test, that the problem doesn't exist in a UP environment, but only in an MP environment. **MPROC** relies upon UP functioning (within reason), and also upon a correctly functioning PDC.

Legal Processor States and Transitions

MPROC allows a processor to be in one (1) of 5 legal states: *MASTER*, *IDLE*, *SLAVE*, *MASTER-TEST*, or *SLAVE-TEST*. To make the diagnostic more robust and have more control over the system, control over these possible states has been given to the user through three commands: **master** n, **slave** n, and **idle** (where n is a single processor number).

State: *MASTER*

The *MASTER* of the diagnostic is the processor that is actually executing the control code and performing I/O. One tends to think of the MONARCH processor as the *MASTER*, but this need only be true upon initialization. The MONARCH processor is the processor that has won arbitration and will thus execute ISL and eventually initiate execution of the diagnostic, whereas the processors who have lost arbitration will be polling for an interrupt (EIR{0}) in PDC's **PDC_RENDEZVOUS** code.

The one requirement for the diagnostic to work properly (and not hose the system) is that there must, at all times, be *one and only one MASTER*. To enforce this the **master** n command behaves in a manner different than that of **idle** or **slave**. **master** n requires that processor n be currently in the *IDLE* state. If this is met, it will "swap the state" of the current master processor with that of the idle processor n. "Swapping the state" means that processor n will pick up execution of the diagnostic *exactly* where the previous *MASTER* was. This is done by saving the register sets, iva table, and program counter (to name a few) of the previous *MASTER* processor (before it was put into the *IDLE* state) and having processor n restore these values into its register sets.

State: *IDLE*

The *IDLE* state consists of a processor polling for an external interrupt (EIR{0}). The register set of a processor in this state is of no interest since it will be supplied a state when it is interrupted (to either a *MASTER* or a *SLAVE*). Upon entry to **MPROC**, all processors that lost arbitration (*MONARCH* selection) are in this state.

State: *SLAVE*

The *SLAVE* state consists of a processor waiting to be released from a semaphore (called **mailbox**). Once it is released, it then will determine which state it is to go to depending upon the information the master has supplied in the **slave_info** data structure:

1. Return to the idle state.
2. Execute a test section (thus going into the *SLAVE-TEST* state).

Only one (1) processor can be in the *SLAVE* state, and it is put into this state by the user typing **slave** n. Notice that it is possible to execute the diagnostic without any processor in the *SLAVE* state. If no *SLAVE* is selected, and **resume** is executed, the diagnostic will automatically sequence through all the selected test sections, ensuring that each processor is a *MASTER* and *SLAVE*. The testing strategy is based on two-way (*MASTER/SINGLE SLAVE*) testing.

State: *SLAVE-TEST*

The *SLAVE* goes here when the *MASTER* has a test to execute. It is the programmer's responsibility to sync up the tests according to the results desired. The method recommended for doing this is by using the semaphores **master_here** and **slave_here**.

The only valid transition from this state is back to the *SLAVE-WAIT* state. This is done by doing a simple return from the procedure (As long as the stack and PCCs are adhered to).

State: *MASTER-TEST*

The *MASTER* enters this state by executing a procedure call. Before doing so, it is the *MASTER*'s responsibility to set the *SLAVE*'s data structure **slave_info** so that the *SLAVE* will be in the correct state. Just before beginning a test (after a test setup), the *MASTER* processor will release the "mailbox" semaphore, allowing the *SLAVE* to enter its next state (*SLAVE-TEST* of *IDLE*).

The general scenario is:

1. Set up *MASTER*
2. Set up *SLAVE*
3. Release *SLAVE* (semaphore)
4. *MASTER* jumps to test section.
5. Optional: sync up the processors again (else one can pretty much assume that the *MASTER* is ahead of the *SLAVE*, due to SMB traffic).
6. Perform *MASTER-TEST* and *SLAVE-TEST*.
7. Put *SLAVE* back into the **slave-wait** state.
8. *MASTER* checks results of the tests and states any errors found along with isolation messages.
9. *MASTER* returns to the **master-wait** state, ready to process next section or next command.

Example: Using the `master`, `slave` and `idle` Commands

Upon entering **MPROC** the state of the processors is: *MASTER* = Proc a, *IDLE* = Proc b, Proc c, and Proc d. Below is an example of how to maneuver control among the processors. Each command is considered to be executed sequentially (*not* independently of the previous one).

To make Proc b the slave:

```
MPROC>  slave b; resume
```

Since all processors are in the *IDLE* state (and therefore no other processor is *SLAVE*) processor b can just be moved from *IDLE* to *SLAVE*.

To make Proc c the next *SLAVE*: MPROC> idle b; slave c; resume

Since processor b was *SLAVE*, the user must first move b into the *IDLE* state (since ONLY 1 processor can be *SLAVE* at one time), and then make c *SLAVE*.

To make Proc c the new *MASTER*:

```
MPROC> idle c; master c; resume
```

Since c is currently *SLAVE*, you must idle it first. Notice that you don't idle Proc a from the *MASTER* state—you simple toggle Proc c's state with Proc a's. The result is Proc c is now executing the code (the **resume** command is actually executed on Proc c) and Proc a is polling for ERI{0} along with Proc b & d.

To run all test sections automatically (assuming that there is no *SLAVE*, but that there is at least one *IDLE* processor), type the following:

```
MPROC>resume
```

All tests are then run, ensuring that each processor functions as both a *MASTER* and a *SLAVE*.

Test Sections

MPROC is responsible for stressing specific SMB transactions.

- Read Shared: (LDW/STW)
- Read Private: (LDW/STW and LDCWS)
- Clear Smart: (LDW/STW and LDCWS)
- Writes to Processor/Cache: (LDCWS instruction for semaphores)
- Coherency Checks - Purge ITLB:
- Coherency Checks - Purge DTLB:
- Coherency Checks - Flush I-Cache:
- Coherency Checks - Flush D-Cache:
- Coherency Checks - Purge D-Cache:

General Algorithms For MP Test Sections

The main task of a multi-processor diagnostic is to verify that the Cache Coherency Check (CCC) is functioning properly. Therefore, the concentration of the diagnostic should be on those instructions which invoke the CCC in an MP environment. These are: `ldw`, `stw`, `fdc`, `pdc`, `fic`, `pitlb`, `pdtlb`, `ldcws`. The instructions which do not cause a CCC are `fice`, `fdce`, `pitlbe`, `pdtlbe`. The basic test sections consist of:

COMBINATIONS OF TESTS TO BE RUN

Test Type	Master	Slave
data cache	LDW/STW	LDW/STW
""	LDW/STW	FDC
""	LDW/STW/LDCWS	FDC
""	LDW	PDC
""	LDCWS	PDC
instr cache	BR	FIC
data TLB	PROBE,LPA	PDTLB
instr TLB	BR*	PITLB

* The execution of a test will cause an ITLB trap

LDW/STW

```
        /* local variable declaration  */
unsigned int cache_array[total data cache size in words];
#define MASTER_WRITE 0X55555555
#define SLAVE_WRITE  0XAAAAAAAA
```

MASTER SET UP:

Load slave_info structure with pc, eiem, etc.

```
release_sem(&mailbox,1);        /* gets slave out of "slave_wait" state */
```

```
Purge_Data_Cache();              /* "for" loop from 0 to max-cache-lines */

release_sem(&master_here,1);     /* sync up with the slave */
get_sem(&slave_here);

  ==============================
```

MASTER TEST:

```
start_timer();                   /* going to check time of master execution vs.
                                    slave execution.  They should be approx.
                                    the same */

for (i=0; i<max_words_in_cache; i++)
{                                /*  The entire cache should be filled with 5's.
                                     NOTE:  Do not overfill cache for it will
                                     mess up timing by including a write to
                                     memory of the cache line that has to be
                                     written out to memory  */
        write_cache(&cache_arry[i],MASTER_WRITE);
}

stop_timer();
master_time = difference_in_times;

release_sem(&master_here,1);     /* sync up with the slave */
get_sem(&slave_here);

==============================
```

MASTER CHECK:

```
This part of the test section determines if ldw is working properly.
```

Note

One assumption that this test strategy is based upon is that the master is in fact ahead of the slave during execution. Therefore, the contents of the cache after this test should, in fact, be all SLAVE_WRITES.

```
Compare_Times();                  /* The slave should be within 60 or so
                                     cycles of the master (if even that much) */

if (difference_in_times > EXPECTED_DIFF)
        { something funny happened...  possible error }

/*  Check Cache_Array for any appearance of SLAVE_WRITES.  */

case (cache_array[i])
{
```

```
        SLAVE_WRITE:    okay, break;

        MASTER_WRITE:   error_routine;
                        master_err_count += 1; /* caution: make sure keep in
                                                  mind local versus global
                                                  errors!  */
                        break;
        default:        real_memory_error      /* possible parity err */
                        exit with as much information.
}

/*   Check Slave's Cache_Array errors:  */
if (slave_err_count > 0) {
        report_error
}
```


```
  ==============================
```

SLAVE SET UP:

```
Purge_Data_Cache();             /* "for" loop from 0 to max-cache-lines */

release_sem(&slave_here,1);     /* sync up with the master */
get_sem(&master_here);
```

```
  ==============================
```

SLAVE TEST:

```
start_timer();                  /* going to check time of master execution vs.
                                   slave execution.  They should be approx.
                                   the same */

for (i=0; i<max_words_in_cache; i++)
{                               /*  The entire cache should be filled with A's.
                                    NOTE:  Do not overfill cache for it will
                                    mess up timing by including a write to
                                    memory of the cache line that has to be
                                    written out to memory  */

        write_cache(&cache_array[i],SLAVE_WRITE);
}

stop_timer();
slave_time = difference_in_times;

release_sem(&slave_here,1);     /* sync up with the master */
get_sem(&master_here);
```

```
  ==============================
```

SLAVE CHECK:

This part of the test section determines if ldw is working properly.

Note

One assumption that this test strategy is based upon is that the master is in fact ahead of the slave during execution. Therefore, the contents of the cache after this test should, in fact, be all SLAVE_WRITES.

```
Compare_Times();                /* The slave should be within 60 or so
                                   cycles of the master (if even that much) */

if (difference_in_times < EXPECTED ){
        something funny happened...  possible error
}

/*  Check Cache_Array for any appearance of SLAVE_WRITES.  */

case (cache_array[i])
{
        SLAVE_WRITE:    okay, break;

        MASTER_WRITE:   error_routine;
                        slave_err_count += 1;       /* caution: make sure
                                                       keep in mind local
                                                       versus global errors */
                        break;
        default:        real memory error (parity).
                        exit with as much information.
}

slave_wait();           /* Return to slave_wait state & wait for next test */
```

==========

Added procedures required:

```
write_cache                                 ; note: there isn't any fdc
        .PROC
        .ENTER
        .CALLINFO NO_CALLS

        ldw gr25,(gr26)

        .EXIT
        .LEAVE
```

FDC

MASTER SET UP:

Load slave_info structure with pc, eiem, etc.

```
release_sem(&mailbox,1);        /* gets slave out of "slave_wait" state */

Purge_Data_Cache();             /* "for" loop from 0 to max-cache-lines */

release_sem(&master_here,1);    /* sync up with the slave */
get_sem(&slave_here);
```

==========

MASTER TEST:

```
write_cache(cache_array[i], MASTER_WRITE)

release_sem(&master_here,1);    /* sync up with the slave */
get_sem(&slave_here);
```

==========

MASTER CHECK:

```
Check_Hardware_Cache_Dirty_bits();    /* they should all be set */
release_sem(&master_here,1);
wait_sem(&slave_here);                /* wait for slave to read cache entries
                                         to verify  */

if (err_cnt != 0) {
      then report errors.
}
```

==========

SLAVE SET UP:

```
Purge_Data_Cache();

release_sem(&slave_here,1);
get_sem(&master_here);
```

==========

SLAVE TEST:

```
get_sem(&master_here);

flush_data_cache(&cache_array[i]);

release_sem(&slave_here,1);
```

```
   ================================
```

SLAVE CHECK:

```
get_sem(&master_here);           /* wait for master to check hw dirty bits. */

Read_Cache (&cache_array[i]) {
        case (cache_word)
        {
                MASTER_WRITE :  OKAY; break;
                0            :  fdc error, err_cnt += 1;
                default      : memory error.
        }
}

release_sem(&slave_here,1);     /* let master continue  */

slave_wait();                   /* return back to slave_wait state */
```

PDC

MASTER SET UP:

Load slave_info structure with pc, eiem, etc.

```
release_sem(&mailbox,1);        /* gets slave out of "slave_wait" state */

Purge_Data_Cache();             /* "for" loop from 0 to max-cache-lines */

release_sem(&master_here,1);    /* sync up with the slave */
get_sem(&slave_here);

  ================================
```

MASTER TEST:

```
write_cache(&cache_array[i],MASTER_WRITE);

/* wait enough so that slave is sure to be done with cache */

purge_data_cache(&cache_array[i]);
release_sem(&master_here,1);
get_sem(&slave_here);          /* wait for slave to finish reading the cache */
  ================================
```

MASTER CHECK:

```
if (err_cnt != 0) {
       report errors
}

  ================================
```

SLAVE SET UP:

```
Purge_Data_Cache();              /* "for" loop from 0 to max-cache-lines */

release_sem(&slave_here,1);      /* sync up with the master */
get_sem(&master_here);
```

==========

SLAVE TEST:

```
write_cache(&cache_array[i],SLAVE_WRITE);

/* wait a while, to be sure purge from master has occurred */
```

==========

SLAVE CHECK:

```
for (whole cache do)
{
        Read_Cache(&cache_array[i]);
        case (cache_array[i])
        {
                MASTER_WRITE :  okay; break
                SLAVE_WRITE  :  error in pdc, err_cnt += 1;
                default      :  memory error!
        }
}
release_sem(&slave_here,1);
get_sem(&master_here);

slave_wait();                   /* return to slave_wait state */
```

LDCWS

This instruction has been simulated to verify that it is atomic. Also, the functionality of the MP Diagnostic System would not work properly if it was not functioning in the correct manner. The real tests for this instruction will be in corner cases. A few situations that should be tested in a similar manner to those tests described above are:

- One processor executing a ldcws while the other is Flushing_Data_Cache.
- One processor executing a ldcws while other is Purging_Data_Cache.
- Try purging the TLB

FIC

This instruction looks at the invalid/dirty bits. The problem occurs as a result of executing out of the same cache. One way to overcome this problem is to fill the cache with "nops" (a carefully coded loop) and then look at it.

PDTLB/PITLB

Check to see if the entries purged from the TLB are marked invalid.

Note

Purge TLB has a special protocol to ensure that any transaction pending in any SPI, completes before the processor that issued the Purge TLB is allowed to process.

HP Part Number
30190-90010 E0391
Edition 6

Printed in U.S.A. March 1991