PA-RISC 1.1 I/O Firmware Architecture Reference Specification

Version 1.0

Printed in U.S.A. August 22, 2001

Notice

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE 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 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.

This page intentionally left blank

4. PDC Procedures

The objective of PDC is to provide a uniform, architected context in which to perform processor-dependent operations. One of the two PDC mechanisms is a software entry point which provides a variety of options to execute specific procedures. These procedures access processor-dependent hardware and return parameters that characterize or identify the processor.

PDC stands for processor-dependent CODE, because the most natural implementation is via Precision code stored in a processor ROM. This does not preclude special hardware support for PDC operations. Any of the PDC procedures may be performed by any combination of code and hardware. In particular, a support processor may be valuable in performing some PDC procedures.

All PDC procedures are provided as options to a single entry point called PDCE_PROC. PDC is a per processor resource, and operating system software must be prepared to manage separate pointers to PDCE_PROC for each processor. The address of PDCE_PROC for the monarch processor is stored in the Page Zero location MEM_PDC. The address of PDCE_PROC for each non-monarch processor is passed in GR26 when PDCE_RESET invokes OS_RENDEZ. The procedure options are selected by the value of the index ARG0 in the call to PDCE_PROC.

ENGINEERING NOTE

A single copy of PDC may be shared between separate Category B processors, provided semaphores or other similar mechanisms are employed within PDC, so that each processor appears to have its own copy.

The calling conventions for PDC procedures are defined in Section 4.1. Section 4.2 contains a table listing all architected PDC procedures. It is followed by a set of pages giving specifications of the PDC procedures. The procedure pages appear in alphabetical order.

Some of the resources used by PDC procedures are system wide and not per processor. This is specified in the respective PDC procedures. When more than one processor module exists on a system, each processor module must return its associated value for per processor resources. For access to system-wide resources, the same value must be returned no matter on which processor the PDC procedure is called.

For all the PDC procedures there is no change in the architecturally visible module state, except where changes have been specified in the respective PDC procedures and calling conventions.

The architected operation of a module (including execution of PDC) must not require the use of any non-architected PDC procedures. It must also not require the use of any non-architected options in architected PDC procedures.

The converse is also true: the execution of a non-architected PDC procedure (or a non-architected option of an architected procedure) must not affect the architected operation of a module.

PDC procedures must not call IODC entry points.

4.1 Calling Conventions

The PDC procedures provide a procedural interface to module-type dependent code for processors. The calling convention used for these procedures is a subcase of the one defined in the *Procedure Calling Convention ACD*, Version 4.1, which allows high level language code to call PDC.

The applicable portions of the calling convention are summarized here; refer to the *Procedure Calling Convention ACD* for details. This subset of the calling convention used by PDC is frozen in the I/O Architecture. Indirect calls through special stubs will be required if the convention changes in the future.

These calling conventions do not apply to the PDC entry points triggered by TOC, reset, or machine checks.

4.1.1 Processor State

The processor must be in the following state when PDC procedures are called:

- The processor must be at Privilege Level 0 at entry, during the PDC procedure call, and at exit.
- The Interruption Vector Table (IVT) is defined by the caller at entry. PDC procedures must not write to the IVT. When the PDC procedure is exited, the IVT must have the same value it had when the procedure was entered.

The following table defines the requirements for the Processor Status Word (PSW) at entry to, during, and at exit from a PDC procedure call. The terms used in the definition of the requirements have the following meaning:

Definition of entry and exit:

Entry — Start of the first instruction of the PDC procedure.

Exit — Start of the first instruction after return to caller's code.

Terms used to define entry and exit values:

0 — Must be set to 0 at entry to or exit from the procedure.

1 — Must be set to 1 at entry to or exit from the procedure.

C — The state of bits marked with C are defined by the caller. If the value at exit is also C, it must be the same as the value at entry.

X — Don't care. On entry, the caller may set these bits to any value. The PDC procedure must not attribute any meaning to the state of these bits. On exit, the PDC procedure may set these bits to any value. The caller must not attribute any meaning to the state of these bits.

Terms used to define allowed actions during execution of the PDC procedure.

Unchanged — The PDC procedure must not change these bits from their entry values at any time during execution of the procedure.

Unspecified — There are no requirements on the state of these bits. The PDC procedure may modify them as necessary during execution of the procedure.

Restored — The PDC procedure may modify these bits as necessary during execution of the procedure, however, they must be restored to their entry value prior to exit from the procedure.

State requirements for the Processor Status Word:

PSW Bit	Entry Value	During PDC Execution	Exit Value	Name
S[1]	C	Unchanged	C	Secure Interval Timer
T	0	Unchanged	0	Taken Branch Trap Enable
H	0	Unchanged	0	Higher Privilege Transfer Trap Enable
L	0	Unchanged	0	Lower Privilege Transfer Trap Enable
N[2]	0	Unspecified	0	Nullify
X[2]	0	Unspecified	0	Data Memory Break Disable
B[2]	0	Unspecified	0	Taken Branch
C	0	Unchanged	0	Code Address Translation Enable
V[3]	X	Unspecified	X	Divide Step Correction
M	0[4]	Restored	0[4]	High Priority Machine Check Mask
C/B[3]	X	Unspecified	X	Carry/Borrow Bits
R	0	Unchanged	0	Recovery Counter Enable
Q	1	Restored	1	Interruption State Collection Enable
P[1]	C	Restored	C	Protection Identifier Validation Enable
D	0	Unchanged	0	Data Address Translation Enable
I	0	Unchanged	0	Ext, P-fail Interrupt, LPMC Unmask
E	0	Unchanged	0	Little-Endian Enable
F	0	Unchanged	0	Performance Coprocessor Enable
G[5]	0	Unchanged	0	Debug Trap Enable
Y[5]	0	Unchanged	0	Data Debug Trap Disable
Z[5]	0	Unchanged	0	Instruction Debug Trap Disable

Notes

1. These bits are Defined by the caller. Their value at exit must be the same at their value at entry.
2. These bits are changed by normal instruction execution. They will be 0 on entry and exit from the PDC procedure. No action is required by the caller.
3. These bits are set by arithmetic instructions during normal execution. No Action is required by the caller.
4. Except when PDC_PIM is called with ARG1=0, in which case the M-bit must be 1.
5. These bits are implemented only on level 0 processors which also implement the debug SFU. In all other cases they are reserved bits.

4.1.2 Register State

The following four tables define the requirements for the State of the Control Registers, Space Registers, General Registers, and Floating Point Registers at entry to, during, and at exit from a PDC procedure call. The terms used in the definition of the requirements have the following meaning:

Terms used to define entry and exit state:

C Defined by the Caller. The value at entry is defined by the caller of the PDC procedure. If the value at exit is also defined by caller, it must be the same as the value at entry.

HV HVERSION (processor) dependent. The value, if it exists, is dependent on the HVERSION of the processor. It must not be used by the PDC procedure or its caller.

X Don't Care. On entry, the caller of the PDC procedure may set the register to any value. The PDC procedure must not attribute any meaning to the value. On exit, the PDC procedure may set the register to any value. The caller must not attribute any meaning to the value.

U The value of the register is not defined for either the PDC procedure on entry or the caller on exit from the procedure.

V The value at exit is a result obtained by execution of the PDC procedure.

The terms used to define allowed actions during execution of the PDC procedure are the same for Registers as for PSW bits, with the following additions:

Set Result — During the PDC procedure call a result is placed in the register prior to exit.

Unused — The PDC procedure is not allowed to write to or rely on the value in the register. However the register may not contain the value at entry during the entire call and at exit.

Register state requirements for Control Registers:

CR No.	Entry Value	During PDC Execution	Exit Value	Name
0	C	Unchanged	C	Recovery Counter
1-7	HV	Undefined	HV	None (HVERSION Dependent)
8	C	Unchanged	C	Protection ID 1
9	C	Unchanged	C	Protection ID 2
10	C	Unchanged/ Set Result[1]	C/V	Coprocessor Configuration Register
11	C	Unspecified	X	Shift Amount Register
12	C	Unchanged	C	Protection ID 3
13	C	Unchanged	C	Protection ID 4
14	C	Unchanged/ Restored[2]	C	Interruption Vector Address
15	C	Restored	C	External Interrupt Enable Mask
16	C	Unused[3]	C[3]	Interval Timer
17	U	Unused	U	Interruption Instruction Address Space Queue
18	U	Unused/ Unspecified[4]	U	Interruption Instruction Address Offset Queue
19	U	Unused	U	Interruption Instruction Register
20	U	Unused	U	Interruption Space Register
21	U	Unused	U	Interruption Offset Register
22	U	Unused/ Unspecified[4]	U	Interruption Processor Status Word
23	C	Unchanged	C	External Interrupt Request Register
24-31	C	Unused/ Restored[5]	C	Interrupt Handler Use

Notes:

1. The only PDC procedure that is allowed to read from or write into the Coprocessor Configuration Register (CR 10) is PDC_COPROC. The exit state of CR 10 is either the value at entry, or the value *ccr_functional* provided by PDC_COPROC.
2. PDC Procedures may write to the IVA (CR 14) only if they handle their own interruptions. The address of the PDC procedure's interruption handlers is written to CR 14. The caller's interruption handler must be restored before returning to the caller.
3. No called PDC procedure is allowed to write to the Interval Timer(CR 16). The value at exit is the value at entry plus the elapsed time of the procedure call.
4. PDC procedures may write to CR 18 and CR 22 when it is necessary for them to modify PSW bits. Only the PSW bits to be changed may be written to CR 22. A Return From Interruption (RFI) instruction must be executed as early as possible following the writes to CR 18 and CR 22 to reduce the probability of the value written to these registers by the PDC procedure being destroyed by an interruption.

PROGRAMMING NOTE

It is recommended that an RFI instruction immediately follow writes to CR 18 and CR 22.

---

5. CR 24 through CR 31 are defined for interruption handler use. If a PDC procedure handles its own interruptions, the handler may use these registers for temporary storage. In any case, PDC must restore the original contents before returning to the caller.

Register state requirements for Space Registers:

SR No.	Entry Value	During PDC Execution	Exit Value
0-2	X	Unspecified	X
3-7	C	Restored	C

Register state requirements for General Registers:

GR No.	Entry Value	During PDC Execution	Exit Value	Usage
0	0	Unchanged	0	Zero
1	X	Unspecified	X	Temporary storage (scratch)
2	C	Unspecified[1]	X	Return Address of caller
3-18	C	Restored	C	Temporary storage (callee saves)
19-22	X	Unspecified	X	Temporary storage (caller saves)
23	C	Unspecified	X	ARG3
24	C	Unspecified	X	ARG2
25	C	Unspecified	X	ARG1
26	C	Unspecified	X	ARG0
27	C	Restored	C	Defined by caller
28	X	Set Result	V	Return Status
29	X	Unspecified	X	Unspecified
30	C	Restored	C	Caller Stack Pointer
31	X	Unspecified	X	Temporary storage (scratch)

Notes:

1. The PDC Procedure may modify GR 2 as necessary during execution, and the caller may not rely on the contents of GR 2 after the call. The PDC procedure must maintain the value passed in GR2 by the caller to use as a return address after the call completes.

Register state requirements for Floating Point Registers:

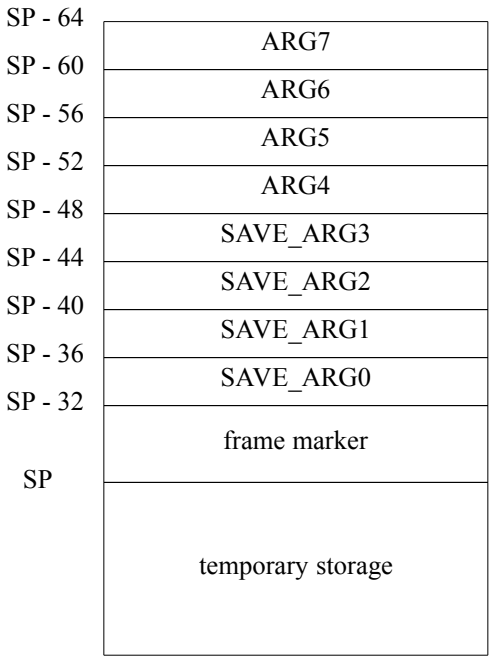
FPR No	Entry Value	During PDC Execution	Exit Value
0	C	Unchanged/[1] Set Result	C/V/HV[1]
1-31	C	Unchanged/[1] Unspecifed	C/HV[1]

Notes:

1. The floating point registers FPR0 through FPR31 are defined by the caller at entry and must be unchanged by all procedures except PDC_COPROC. When PDC_COPROC exits, the values of FPR 0 through FPR 31 are all HVERSION dependent except for the T bit of FPR 0. The T bit of FPR0 returns the test status and must be set to 0 if the test succeeds.

4.1.3 Stack Usage

The caller of PDC procedures must provide a doubleword-aligned value in GR 30, the Stack Pointer (SP), which points to the following data:



The values of SAVE_ARG0 through SAVE_ARG3 are defined by the caller at entry, and are unspecified at exit. They are used to save the first four arguments to the procedure, which are passed to the procedure in registers GR26 through GR23, respectively. Additional arguments are passed on the stack in successive locations (ARG8 at SP-68, ARG9 at SP-72, ARG10 at SP-76, etc.)

When the called procedure returns, the value of SP must be restored. The memory at the Stack Pointer address SP and the next 7K bytes of larger physical addresses are available for temporary use by the called procedure. If the called procedure is PDC_POW_FAIL, the memory at the Stack Pointer address SP and the next 512 bytes of larger physical addresses are available for use by PDC_POW_FAIL.

4.1.4 Arguments

The procedure to be performed is selected by the index **ARG0**. ARG0 is not explicitly listed in the procedure specifications that follow. The ARG0 value for all PDC procedures is a 32-bit unsigned integer.

The option of the procedure is selected by **ARG1**. For architected PDC procedures, options 0 through 127 are architected or reserved; the remaining options (128 through X'FFFFFFFF) are for HVERSION-dependent use. For HVERSION-dependent PDC procedures, all options are for HVERSION-dependent use. The ARG1 value for all PDC procedures is a 32-bit unsigned integer.

Many PDC procedures use the standard argument *R_addr* to designate the return parameter buffer. This buffer is a doubleword-aligned block of 32 words allocated by the caller. The procedure can return parameters to its caller by storing into the buffer. If the *R_addr* argument is provided, it is always **ARG2**.

In the procedure specifications that follow, the notation 'R' is used to indicate an argument passed to a PDC procedure which is reserved for future extensions. Reserved arguments must be set by all current callers to 0, and must be ignored by all current callees. Reserved arguments may be architected in the future, with the value 0 defined to preserve compatibility with previous versions.

The notation 'HV' is used to indicate that the value of the argument is not specified by the architecture and so may be freely chosen by the caller. By contrast, arguments denoted by '---' are nonexistent: the caller is not required to provide such arguments at all. Callees must not attach any significance to 'HV' arguments and must not attempt to access '---' arguments.

All address parameters which are passed as arguments or are returned as parameters are 32-bit unsigned integers. The alignment restrictions for these address parameters are specified in each of the PDC procedures.

All signed integers are represented in two's complement (32-bit) format.

4.1.5 Return Parameters

If a PDC procedure returns parameters to its caller, they are stored in the return parameter buffer specified by *R_addr*. The 32 returned parameters are called RET[0] through RET[31]. At least RET[0] through RET[15] are designated for architected return parameters. Return parameters in RET[16] through RET[31] which are not architecturally defined may be used for HVERSION-dependent purposes. All return parameters neither architected nor used for HVERSION-dependent purposes must be set to 0 by the PDC procedure upon return. The notation 'R' indicates a return value that must be set to 0 by the PDC procedure. The notation '---' indicates that the PDC procedure does not return any parameters.

If a PDC implementation defines a new dependent return word for a procedure, the value 0 must be used to indicate "not implemented" to preserve compatibility with previous versions.

For HVERSION-dependent PDC procedures and HVERSION-dependent options of architected PDC procedures, all 32 return values (RET[0] through RET[31]) are HVERSION dependent.

4.1.6 Status

The status of PDC procedures is returned as a 32-bit two's complement signed integer value in register GR28.

The rest of this section applies only to architected options of architected PDC procedures. For HVERSION-dependent PDC procedures and HVERSION-dependent options of architected PDC procedures, all status values are HVERSION dependent.

The following status values have the same meaning for all PDC procedures to which they apply:

Value	Description
3	Call completed with a warning
0	OK
-1	Nonexistent procedure
-2	Nonexistent option
-3	Cannot complete call without error
-10	Invalid argument
-12	BUS_POW_WARN assertion detected

Note that status -1 is actually returned by PDCE_PROC, the entry point for all PDC procedures, rather than by the individual PDC procedures themselves.

Positive status values (values from 1 to X'7FFFFFFF) are used to report advisory information whose meaning is dependent on the procedure that was called.

The other negative status values (values from -4 to -9 and -11 to -X'80000000) are used for errors whose meaning is dependent on the procedure that was called.

Status values other than those listed for a PDC procedure are reserved. Each PDC procedure may return only the values specifically defined for it. Reserved values can be assigned architected meanings in the future. Therefore, callers must treat the reserved negative values the same as -3 (Cannot complete call without error) and the reserved positive values the same as 0 (OK).

Some status values are marked as REQUIRED. This means that all implementations of the PDC procedure are required to detect the condition specified by the status value and to return the status value whenever the condition is detected. Values are designated as required when necessary to support the functionality of the procedure.

Some status values are marked as OPTIONAL. This means that each implementation of the PDC procedure can choose whether or not it will detect the condition specified by the status value.

Some status values are marked as CONDITIONAL. These values are accompanied by a specification of the cases in which the condition must be detected and reported. There will be some PDC implementations for which those

cases do not apply; they must not use the given value at all.

ENGINEERING NOTE

It is expected that those PDC implementations that are able to detect optional conditions will do so (and will return the appropriate status value).

PDC implementations are encouraged to recognize as many specific error conditions as they can.

If an implementation cannot isolate an error to one of the more specific conditions, then it must report the error by returning the general status value -3 (indicating that an indeterminate error was detected). If it cannot isolate one of the specific advisory conditions, then it must return status 0 for "OK".

For all PDC procedures:

- All RET values are valid with a zero return status.
- All RET values are valid with any positive return status unless specified otherwise in the respective PDC procedure description.
- All RET values are HVERSION dependent with a negative return status.

4.1.7 Interruptions

PDC procedures may optionally handle their own interruptions.

If PDC procedures do not handle their own interruptions, the execution of PDC procedures must not cause any Group 3 or Group 4 interruptions. If PDC procedures do not handle their own interruptions, they must not write to CR14 or to CR24 through CR31 during the procedure call.

If PDC procedures handle their own interruptions, they must write to CR 14 during the procedure call to point to their interruption handlers. They may optionally modify the contents of CR 24-31 in accordance with their interruption handlers. The interruption handlers must not modify those registers which the PDC procedures are not allowed to modify.

PDC procedures are not required to recover if they receive an HPMC.

ENGINEERING NOTE

It is recommended that all PDC procedures be re-entrant to help them recover from an HPMC. The probability of receiving an HPMC during a PDC procedure call is considered sufficiently low that PDC procedures are not required to be re-entrant.

4.1.8 Powerfail Warning on Central Bus during PDC Call

In the event of a powerfail warning on the central bus while a PDC call is in progress, the procedure must always return quickly enough so that the caller has the full powerfail budget available. PDC_CHASSIS and PDC_POW_FAIL must complete all required actions before returning. All PDC procedures except PDC_CHASSIS and PDC_POW_FAIL have two choices in the event of a powerfail warning on the central bus:

- complete all required actions before returning with the appropriate status
- terminate the call prematurely and return -12

A PDC procedure that returns -12 must be restartable, that is, it can be re-executed by the caller.

PROGRAMMING NOTE

The recommended calling sequence for restartable PDC procedures follows:

```
while ( 1 ) {
     PSW I—bit ← 0;
     status ← PDC_xxxxx ( <arguments> );
     if (status != —12)
          break;
     PSW I—bit ← 1;
}
```

4.2 PDC Procedures

When PDCE_PROC is called, the index ARG0 specifies the procedure to be performed, as shown below:

ARG0	Mode	Name	Description
0	R		Obsolete
1	HV	PDC_POW_FAIL	Prepare for powerfail
2	A	PDC_CHASSIS	Update chassis display
3	A	PDC_PIM	Access Processor Internal Memory
4	A	PDC_MODEL	Return processor model information
5	A	PDC_CACHE	Return cache and TLB parameters
6	A	PDC_HPA	Return processor's HPA
7	A	PDC_COPROC	Return coprocessor configuration
8	A	PDC_IODC	Access a module's IODC
9	A	PDC_TOD	Access Time-Of-Day clock
10	A	PDC_STABLE	Access Stable Storage
11	HV[1]	PDC_NVOLATILE	Access Non-Volatile Memory
12	A	PDC_ADD_VALID	Validate address
13	R		Obsolete
14	R		Obsolete
15	R		Obsolete
16	A[3]	PDC_PROC	Stop the currently executing processor
17	HV[4]	PDC_CONFIG	Deconfigure and reconfigure a module
18	R		Obsolete
19	HV[5]	PDC_TLB	Manage hardware TLB miss handling
20	HV	PDC_MEM	Manage per page memory deallocation
21	HV[6]	PDC_PSW	Manage default PSW bits
22	HV	PDC_SYSTEM_MAP	Map fixed location IO modules
23	HV	PDC_SOFT_POWER	Provide software controlled power off
24	HV[7]	PDC_ALLOC	Allocate IODC data area
25-127	R		Reserved
128-511	HV		HVERSION dependent
> 511	R		Reserved

R	the procedure index is Reserved
A	the procedure is required by the architecture in all PDC implementations
HV	the existence of the procedure depends on the HVERSION of the processor

Notes:

1. Required for processors that provide Non-Volatile Memory; not implemented by other processors.
2. Required for processors that immediately enter the Interruption Vector Table upon an HPMC fault condition. Processors that trigger PDCE_CHECK directly upon a machine check fault condition do not implement this procedure.
3. Required for Category B processors; not implemented by Category A processors.
4. Required for processors that support module deconfiguration; not implemented by other processors.
5. Required for processors with hardware TLB miss handlers; not implemented by other processors.
6. Required for processors which implement the E-bit or W-Bit; not implemented by other processors.
7. Required for processors whose IODC firmware required data area allocation; not required for other processors.

If an implementation provides a PDC procedure, then it must provide all defined options for that procedure, unless explicitly noted to the contrary under the "**Options**" heading in the specifications that follow.

PDC_ADD_VALID (index 12)

Purpose: To determine the correct completion of a READ operation without risking an HPMC.

Arguments:

Description	ARG1	ARG2
Validate address	0	phaddr

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
2	Operation completed with a requestor bus error REQUIRED.
1	Operation completed with a requestor bus error REQUIRED.
0	OK (Operation completed without bus error) The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: The "**Validate address**" option (ARG1=0) allows the caller to determine if a READ to *phaddr* would generate a requestor bus error without risking an HPMC. PDC_ADD_VALID generates a READ operation by issuing a load instruction to *phaddr*. The *phaddr* argument is a word-aligned, 32-bit physical address.

PDC_ADD_VALID must return either 1 or 2 when detecting an HPMC fault condition caused by a requestor bus error. PDC_ADD_VALID may return either status value, so the caller must check for both values. It must also return one of these values when called for a deconfigured molule which does not normally assert PATH_SLAVE_ACK in that state.

For processors that do not implement the full 32-bit address space, PDC_ADD_VALID must perform a simple address comparison to check that *phaddr* is within the implemented address space. PDC_ADD_VALID must not try to determine the validity of a given physical address by attempting a READ before this test is successful. If *phaddr* is in the unimplemented portion of the address space, PDC_ADD_VALID must return either 1 or 2.

If PDC_ADD_VALID detects an HPMC condition other than a requestor bus error, it is required to return one of the following status values: 2, 1, 0, or -3. Status value 0 indicates that the HPMC condition did not interfere with the completion of the READ operation. Status value -3 indicates that the HPMC condition interfered with the completion of the READ operation (that is: PDC_ADD_VALID could not determine that the read to *phaddr* completed). Status values 1 or 2 may optionally be used when an HPMC condition occurs other than a requestor bus error.

PDC_ADD_VALID may optionally enter its caller's HPMC handler when detecting an HPMC condition other than a requestor bus error.

SUPPORT NOTE

Implementations of PDC_ADD VALID are encouraged to use status values 0 or -3 for HPMC condtions other than requestor bus errors. Support organizations will decide whether or not an implementation is allowed to use the status values 2 or 1 for these HPMC conditions. This decision is based on the supportability, reliability, and availability requirements of the particular product.

PROGRAMMING NOTE

Calls to PDC_ADD_VALID may cause soft errors to be logged in the lower ports of bus converters involved in the READ operation. Therefore, the caller may wish to issue a CMD_CLEAR to any bus converter ports that may have been affected. Issuing CMD_CLEAR is optional, but future error isolation is hampered by having bus converter ports with residual soft errors logged during prior calls to PDC_ADD_VALID.

PDC_ALLOC (index 24)

Purpose: To allocate static data storage for IODC use.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4
Allocate Storage	0	R_addr	Size	R

Returns:

Description	RET[0]
Allocate Storage	Storage_address

Status:

Value	Description
1	Storage requrest not alligned to cache-line boundary. Storage returned was rounded up to the next larger cach-line boundary. REQUIRED.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-4	Insufficient Storage There was not enough storage available to satify the request. REQUIRED.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: The "**Allocate Storage**" option (ARG1=0) is used to allocate static storage for use by IODC. Once allocated, the storage can not be freed, therefor it's use should be limited to once per boot for a particular user. The single exception to this rule is that boot storage may be freed by zeroing the architected locations in page zero. The *Size argument* denotes the size of the allocation requested. There is a total of 32K of storage available for allocation. The address of the storage area allocated is returned in the *Storage_address* return parameter.

The allocation request should be in cache line increments. The storage returned will be cache line aligned.

PDC_CACHE (index 5)

Purpose: To return the cache and TLB configuration parameters and to set the cache coherence state.

It is expected that the operating system will call PDC_CACHE during system configuration. This enables the operating system to use the cache and TLB more efficiently.

Options: Option ARG1=1 must be implemented in Category B processors which issue non-coherent operations instead of coherent operations during the execution of PDCE_CHECK and OS_HPMC. See the PDCE_CHECK description in Section 3.2, PDC Entry Points.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4	ARG5	ARG6	ARG7
Return parameters	0	R_addr	HV	R	R	R	R
Set coherence state	1	R_addr	Is_cst	Ds_cst	ITs_cst	DTs_cst	R
Return space-ID bits	2	R_addr	R	R	R	R	R

Returns: Returns for ARG1=0:

RET[0] I_size	RET[1] I_conf	RET[2] I_base	RET[3] I_stride	RET[4] I_count	RET[5] I_loop
RET[6] D_size	RET[7] D_conf	RET[8] D_base	RET[9] D_stride	RET[10] D_count	RET[11] D_loop
RET[12] IT_size	RET[13] IT_conf	RET[14] IT_sp_base	RET[15] IT_sp_stride	RET[16] IT_sp_count	RET[17] IT_off_base
RET[18] IT_off_stride	RET[19] IT_off_count	RET[20] IT_loop			
RET[21] DT_size	RET[22] DT_conf	RET[23] DT_sp_base	RET[24] DT_sp_stride	RET[25] DT_sp_count	RET[26] DT_off_base
RET[27] DT_off_stride	RET[28] DT_off_count	RET[29] DT_loop			

Returns for ARG1=1:

RET[0] Ia_cst	RET[1] Da_cst	RET[2] ITa_cst	RET[3] DTa_cst

Returns for ARG1=2:

RET[0]
Space_bits

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
1	Error detected, partial use possible Some test failed. The returned parameters are for the usable (as opposed to the manufactured) configuration. Some coherence states not changed to requested value. CONDITIONAL. Must be used if partial use after error is possible.

0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: The "**Return parameters**" option (ARG1=0) returns 30 parameters that characterize the processor's caches and TLBs. All *_size*, *_base*, *_stride*, *_count*, and *_loop* parameters are 32-bit unsigned integers. The returned parameters reflect the configuration determined to be usable by optional tests run during or prior to the PDC_CACHE call.

The D-cache parameters (RET[6] - RET[11]) are not necessarily duplicates of the I-cache parameters (RET[0] - RET[5]). As well, The DTLB parameters (RET[21] - RET[29]) are not necessarily duplicates of the ITLB parameters (RET[12] - RET[20]). Software must read each set of parameters.

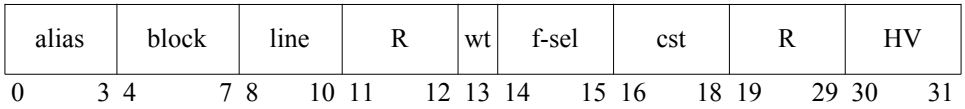
If a machine check causes caches or TLBs to be reconfigured, PDC_CACHE must continue to return the same cache and TLB parameters.

Data Cache Parameters

D_size specifies the size of the D-cache in bytes, exclusive of tags and other descriptors. It is calculated as the effective size of all the levels of the data cache and the combined cache, when the cache system is fully configured (error conditions may reduce the size of an operational cache).

If *D_size* is 0, then there is no D-cache and *D_conf*, *D_base*, *D_stride*, *D_count*, and *D_loop* must also be 0.

D_conf specifies the configuration of the D-cache, in the following format:



The *alias* field specifies the aliasing boundaries for virtual addresses. The values returned are defined as follows:

Value	Description
0	Unknown[1]
1	4 KB
2	8 KB
3	16 KB
4	32 KB
5	64 KB

6	128KB
7	256 KB
8	512 KB
9	1 MB
10	2 MB
11	4 MB
12	8 MB
13	16 MB
14-15	Reserved

1. The aliasing boundary is unknown and may be greater than 16MB.

The *line* field specifies the maximum amount of data that will be written back to memory as the result of a store instruction, expressed as a multiple of 16 bytes. This data written to memory is aligned to an address which is a multiple of the line size. The allowed values of *line* are 0 (if no cache), 1, 2, and 4.

PROGRAMMING NOTE

Software can use the value of the D-cache line size in assigning memory addresses that can be modified by other modules. To avoid generating indeterminate data, other modules should be prevented from modifying a memory location contained in a D-cache line. It is also possible to use semaphores to control the access to memory that is shared by a processor and another module.

For alignment purposes, software can always assume a value of 64 bytes as the D-cache line size, because it is the maximum value allowed by the architecture.

The value of *block* can be used to determine the most efficient stride for use by software to flush or purge a range of addresses. The value of *block* is 0 only if the D-cache is not implemented. The value of this stride is given by $2^{block-1} * line * 16$. A flush or purge of an address will flush or purge the aligned data block of size $2^{block-1} * line * 16$. (The entire data block will be ejected from the D-cache; only the lines that are dirty will be written to memory.)

PROGRAMMING NOTE

Smaller address strides can also be used in flushing or purging. In fact, the value 16 can always be assumed for the address stride for flush and purge instructions.

The value of *wt* is 0 if the D-cache is a write-back cache, and is 1 if the D-cache is a write-through cache.

If this bit is a 1, it means that any processor store (or semaphore) instruction is architecturally equivalent to the instruction sequence shown below. Any cache states that the cache can end up in as a result of the equivalent instruction sequence are legal.

```
store (or ldcw);
flush;
sync;
load;
```

ENGINEERING NOTE

For a write-back cache, it is typical to have one 'dirty' bit for each line in the D-cache. The 'dirty' bit is set if the line contains data that was stored to but not written to memory. After a store instruction writes into any part of a D-cache line (making it

dirty), the entire line will be written to memory before it is replaced or removed.

PROGRAMMING NOTE

If a processor has a write-through D-cache, the cache contents do not need to be flushed before DMA is initiated nor during powerfail preparation, but a SYNC instruction is still necessary.

The *f-sel* field tells software how to flush a range of addresses from the cache and has the following meaning.

Value	Description
00	Both FIC and FDC must be used
01	Only need FDC
10	Only need FIC
11	Either FIC or FDC may be used

However, if the page was accessed either as instructions only or as data only, then either a FIC loop alone (or the FICE loop), or a FDC loop alone (or the FDCE loop) may be used to flush a range of addresses (or the entire cache). In multiprocessor systems, software must look at all the *f-sel* fields and flush in such a way (either one of the set of flushes, or both) that the address range is flushed on all the processors. The *f-sel* fields of both D_conf and I_conf must be identical.

Independent of the *f-sel* field, software has to execute both the FDCE and the FICE loops to flush either the I-cache, the D-cache, or the entire cache system. However, if an address range has been accessed either as instructions only, or as data only, then either the FICE loop, or the FDCE loop alone may be used to flush the address range.

A value of 0 in the *cst* field means that the D-cache is not issuing coherent operations; a value of 1 means that the D-cache is issuing coherent operations. Values 2 through 7 are reserved. The *cst* field must always be 0 on category A processors.

The four parameters *D_base*, *D_stride*, *D_count*, and *D_loop* are provided by PDC_CACHE for use by software that desires to flush the entire D-cache in an efficient manner. The meaning of these four parameters will be described first from the perspective of the software that uses them and then in terms of the responsibilities of the PDC implementation that must provide them.

The four parameters have meaning only within the context of the given procedure which flushes the entire D-cache. The caller of PDC_CACHE may not assume any other meaning.

Perspective of the PDC_CACHE caller

It is always possible to flush the entire D-cache by using the FDC instruction for every physical address from 0 to the largest possible memory address (X'EEFFFFFF) and for every virtual address from 0 to the largest possible virtual address. Software can accomplish this flushing more efficiently by using the FDCE instruction, but only if it follows all the rules below:

- Software must use an equivalent of the C routine given below (where *D_base*, *D_count*, *D_loop*, and *D_stride* are the parameters returned by PDC_CACHE).

```
unsigned int addr,count,loop,D_base,D_count,D_loop,D_stride;
addr ← D_base;
for (count ← 0; count < D_count; count++) {
    for (loop ← 0; loop < D_loop; loop++)
        FDCE(addr);
    addr ← addr + D_stride;
}
```

A routine is considered equivalent to the C routine if it generates the same sequence of FDCE

instructions.

- During the execution of this loop (including trap handling), memory management instructions, loads, stores, and load and clear word instructions must not be executed.
- Software must hold the space bits constant during the execution of the entire loop, even if the addresses generated by the loop differ in the two most significant bits of the offset.
- All the FDCE instructions in the data cache flush loop must execute with the same value in the PSW D-bit.

The parameters can also be used to optimize the flushing of a range of addresses. The following programming note demonstrates this use.

PROGRAMMING NOTE

The following routine flushes the range of addresses between (space, off_base) and (space, off_bound):

```
unsigned int off_base, off_bound, offset;
for (offset=off_base; offset<=off_bound; offset+=pow(2,block–1)*line*16)
  FDC(space, offset);
FDC(space, off_bound);              /*this is required if
                                      the starting address is not
                                      block–aligned*/
```

This routine must execute with the PSW D-bit equal to 1 if the range to be flushed is a virtual address range, and with the D-bit equal to 0 if the range is a physical range. In a multiprocessor system where the individual processors flush different amounts of the cache on a flush, the minimum value of the product $2^{block-1}$*line*16 should be used in the address range cache flush loop.

Implementors are encouraged to report the product $2^{block-1}$*line*16 to be greater than or equal to the coherence size of the system because software that is flushing a range of addresses would perform redundant flushes (hardware is built to flush a block of the cache = coherence size upon each flush, but software is issuing flushes on a smaller granularity).

Responsibilities of the PDC_CACHE implementation

The designers that specified how the FDCE instruction is implemented and how the D-cache is organized are responsible for identifying values of the four parameters so as to fulfill the promise to software made above.

Space register bits may be used in the hash used to index cache. Although space bits are not varied in the cache flush routine, the entire cache must still be flushed by executing the loop.

The cache flush loop must also work in the presence of coherent cache operations (coherence checks, and broadcast FIC, FDC, PDC, PDTLB, and PITLB operations).

ENGINEERING NOTE

This is the model of cache organization used in defining the C routine. It is included to guide cache designers in identifying values for the four parameters. This is not intended to restrict designers in any way. For example, it is allowed for a processor to flush the entire cache in response to a single FDCE instruction.

D_base is used to establish the starting address. It will usually be possible to set it to 0 with no loss of generality.

D_stride will usually be the size of a cache line in bytes. Thus, one would expect its value to be the *D_conf*[line] field multiplied by 16.

D_count will usually be the number of lines in the cache.

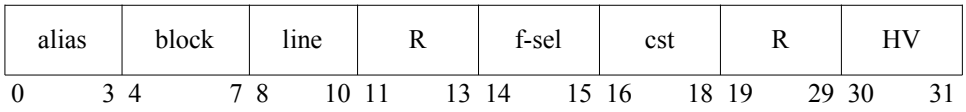
D_loop is intended for set-associative caches. It is used to force the FDCE instruction to be executed multiple times with the same address. Note that when *D_loop* = 1, software can optimize out the inner loop of the C routine. So when there are multiple sets of parameters that all get the flushing job done, the one with *D_loop* = 1 may be most efficient. Implementations that flush all elements of an associative set with a single FDCE instruction will probably use *D_loop* = 1.

Instruction Cache Parameters

I_size specifies the size of the I-cache in bytes, exclusive of tags and other descriptors. It is calculated as the effective size of all the levels of the instruction cache and the combined cache, when the cache system is fully configured (error conditions may reduce the size of an operational cache).

If *I_size* is 0, then there is no I-cache and *I_conf*, *I_base*, *I_stride*, *I_count*, and *I_loop* must also be 0.

I_conf specifies the configuration of the I-cache, in the following format:



The value of *line* has no meaning by itself, because instructions cannot become dirty and be written back to memory. The most efficient stride for flushing a range of addresses from the I-cache is given by $2^{block-1} * line * 16$. The value of *block* is 0 only if the I-cache is not implemented. The *f-sel* field of *I_conf* must be identical to that of *D_conf* word. The *alias* field of *I_conf* returns the offset aliasing boundary for virtual addresses as explained in the *D_conf* description, but may have a different value if the instructions and data caches do not have the same aliasing boundary.

A value of 0 in the *cst* field means that the I-cache is not issuing coherent operations; a value of 1 means that the I-cache is issuing coherent operations. Values 2 through 7 are reserved.

The four parameters *I_base*, *I_stride*, *I_count*, and *I_loop* are provided by PDC_CACHE for use by software that desires to flush the entire I-cache in an efficient manner. The four parameters have meaning only within the context of the given procedure which flushes the entire I-cache. The caller of PDC_CACHE may not assume any other meaning.

The four parameters *I_base*, *I_stride*, *I_count*, and *I_loop* are used in a procedure which is guaranteed to flush the entire I-cache. Software can accomplish this flushing by using the FICE instruction, but only if it follows the rules below:

- Software must use an equivalent of the C routine given below (where *I_base*, *I_count*, *I_loop*, and *I_stride* are the parameters returned by PDC_CACHE).

```
unsigned int addr,count,loop,I_base,I_count,I_loop,I_stride;
addr ← I_base;
for (count ← 0; count < I_count; count++) {
    for (loop ← 0; loop < I_loop; loop++)
        FICE(addr);
    addr ← addr + I_stride;
}
```

  A routine is considered equivalent to the C routine if it generates the same sequence of FICE instructions.

- Software need not insure that there are no extraneous interactions with the I-cache while the routine is being executed. It would not be possible to meet such a condition, because the code is executing out of the I-cache. During the execution of this loop (including trap handling), memory management instructions must not be executed.
- Software must hold the space bits constant during the execution of the entire loop, even if the addresses generated by the loop cross quadrant boundaries.
- All the FICE instructions in the loop must execute with the same value in the PSW D- and C-bits.

Space register bits may be used in the hash used to index cache. Although there is no variation of space in the cache flush routine, the entire cache must still be flushed by executing the loop.

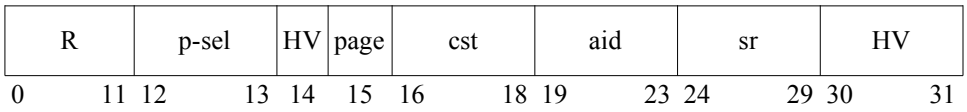
The cache flush loop must also work in the presence of coherent cache operations (coherence checks, and broadcast FIC, FDC, PDC, PDTLB, and PITLB operations).

Instruction TLB Parameters

IT_size specifies the maximum number of entries in the instruction TLB which is calculated as the effective size of all the levels of the ITLB and combined TLBs in a fully configured situation (no entries have been locked out, or deconfigured).

If *IT_size* is 0, then there is no instruction TLB and *IT_conf*, *IT_sp_base*, *IT_sp_stride*, *IT_sp_count*, *IT_off_base*, *IT_off_stride*, *IT_off_count*, and *IT_loop* must also be 0.

IT_conf specifies the configuration of the instruction TLB, as follows:



The *p-sel* field tells software how to purge the TLBs and has the following meaning:

Value	Description
00	Both PITLB and PDTLB must be used
01	Only need PDTLB
10	Only need PITLB
11	Either PITLB or PDTLB may be used

In multiprocessor systems, software must look at all the *p-sel* fields and purge in such a way (either one of the set of purges, or both) that the translations are purged on all the processors.

The architectural page size is 4 Kbytes. Some machines, however, implemented 2 Kbyte pages. The *page* field can be used to identify 2 Kbyte page machines. *page* is normally 1, but is 0 for machines with 2 Kbyte pages.

PROGRAMMING NOTE

Software should be aware that some Precision systems have a 2-Kbyte page size and a 2-Kbyte alignment restriction.

A value of 0 in the *cst* field means that the ITLB is not issuing coherent operations; a value of 1 means that the TLB is issuing coherent operations. Values 2 through 7 are reserved.

The width of the access ids of the processor is encoded in the *aid* field. The width is 15 + *aid*. The width of the space registers is encoded in the *sr* field. If the processor is Level 2, the *sr* field specifies the number of additional spare register bits beyond 32 which are available. The *sr* field is reserved for non Level 2 processors, and the width of the space registers is determined directly

by the Level (0, 16, or 24 for Level 0, Level 1, or Level 1.5, respectively)

The seven parameters *IT_sp_base*, *IT_sp_stride*, *IT_sp_count*, *IT_off_base*, *IT_off_stride*, *IT_off_count*, and *IT_loop* are used in a procedure which is guaranteed to purge the entire instruction TLB. The parameters make the purging procedure work but have no other meaning. The procedure must be run in an environment in which no extraneous TLB interactions can occur (which can be assured if the routine runs with the PSW C- and D-bits=0 and with external interrupts masked). The following C routine (using the values returned by PDC_CACHE) is guaranteed to purge the entire instruction TLB:

```
unsigned int space, sp_count, IT_sp_base, IT_sp_stride, IT_sp_count,
    offset, off_count, IT_off_base, IT_off_stride, IT_off_count,
    loop, IT_loop;
space ← IT_sp_base;
for (sp_count ← 0; sp_count < IT_sp_count; sp_count++) {
    offset ← IT_off_base;
    for (off_count ← 0; off_count < IT_off_count; off_count++) {
        for (loop ← 0; loop < IT_loop; loop++)
            PITLBE(space, offset);
        offset ← offset + IT_off_stride;
    }
    space ← space + IT_sp_stride;
  }
```

ENGINEERING NOTE

This is the model of TLB organization used in defining the C routine. It is included to guide TLB designers in identifying values for the seven parameters. The model assumes that a TLB entry is accessed by hashing together portions of the space and offset of the address. The two outer loops are used to generate the combinations of spaces and offsets that amongst them hash to every entry in the TLB.

IT_sp_base is used to establish the starting space. It will usually be possible to set it to 0 with no loss of generality.

IT_sp_stride is the increment to the space part of the address. If the hashing function does not use the N least significant bits of the space, then IT_sp_stride would be 2^N. This rule of thumb remains valid even if the hashing function does use the least significant bits of the space (N = 0 implies *IT_sp_stride* = 1).

IT_sp_count is the number of space values that must be generated. If the hashing function uses M contiguous bits in the space, then *IT_sp_count* need not be greater than 2^M. This rule of thumb remains valid even if the hashing function does not use the space at all (M = 0 implies *IT_sp_count* = 1).

IT_off_base is used to establish the starting offset. It will usually be possible to set it to 0 with no loss of generality.

IT_off_stride is the increment to the offset part of the address. If the hashing function does not use the P least significant bits of the offset, then *IT_off_stride* would be 2^P.

IT_off_count is the number of offset values that must be generated. If the hashing function uses Q contiguous bits in the offset, *IT_off_count* need not exceed 2^Q.

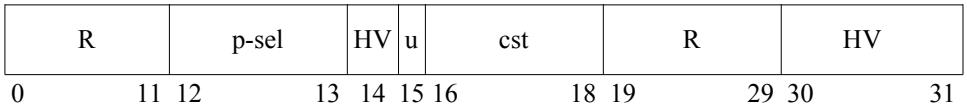
IT_loop is intended for set-associative TLBs. It is used to force the PITLBE instruction to be executed multiple times with the same address. Note that, when *IT_loop* = 1, software can optimize out the innermost loop of the C routine. So when there are multiple sets of parameters that all get the purging job done, the one with *IT_loop* = 1 may be most efficient.

Data TLB Parameters

DT_size specifies the maximum number of entries in the data TLB which is calculated as the effective size of all the levels of the DTLB and combined TLBs in a fully configured situation (no entries have been locked out or deconfigured).

If *DT_size* is 0, then there is no data TLB and *DT_conf*, *DT_sp_base*, *DT_sp_stride*, *DT_sp_count*, *DT_off_base*, *DT_off_stride*, *DT_off_count*, and *DT_loop* must also be 0.

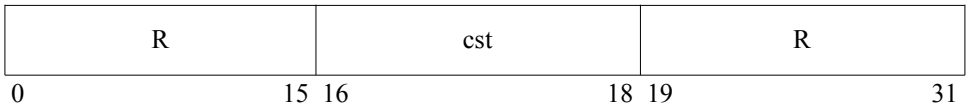
DT_conf specifies the configuration of the data TLB, as follows:



The meanings of the fields within *DT_conf* are analogous to the corresponding fields of *IT_conf* except for bit 15, which is the *u* field instead of the *page* field, and bits 19-29 which are Reserved. A 1 in the *u* field indicates that the processor implements the TLB u-bit, and a 0 indicates that it does not.

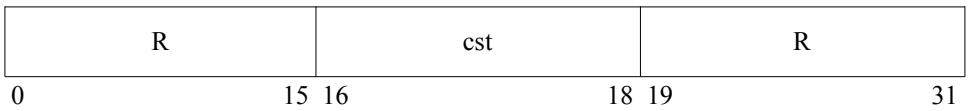
The seven parameters *DT_sp_base*, *DT_sp_stride*, *DT_sp_count*, *DT_off_base*, *DT_off_stride*, *DT_off_count*, and *DT_loop* can be used to purge the entire data TLB. The C routine which purges the entire data TLB is analogous to the one given to purge the entire instruction TLB.

The "**Set coherence state**" option (ARG1=1) attempts to set the current coherence state in the I- and D-caches and TLBs. The following is the format for *Is_cst*, *Ds_cst*, *ITs_cst*, and *DTs_cst*:



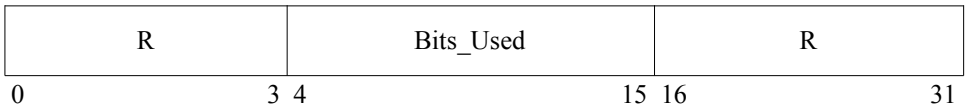
The *cst* field is the coherence state desired for the I-cache, D-cache, ITLB, and DTLB. A value of 0 in the *cst* field means that the cache/TLB do not issue coherent operations; a value of 1 means that the cache/TLB do issue coherent operations. Values 2 through 7 are reserved.

Return parameters *Ia_cst*, *Da_cst*, *ITa_cst*, and *DTa_cst* indicate the actual coherence state. The following is the format for Ia_cst, Da_cst, ITa_cst, and DTa_cst:



If a return value differs from its corresponding argument, then the processor is unable to change to the desired state, and the return value is the current, unchanged state.

The "**Return space-ID bits**" option (ARG1=2) returns whether space-ID hashing is turned on, and which bits are used in the hashing algorithm. If space-ID hashing is not turned on, a 0 will be returned in the *Space_bits* parameter. If space-ID hashing is turned on, the return will be non-zero, and will be formatted as follows:



PDC_CHASSIS (index 2)

Purpose: To update the chassis display and return chassis warnings.

Arguments:

Description	ARG1	ARG2	ARG3
Update chassis display	0	data	HV
Return chassis warnings	1	R_addr	HV
Update display and return warnings	2	R_addr	data

Returns:

Description	RET[0]
Update chassis display	---
Return chassis warnings	warn
Update display and return warnings	warn

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.

Description: Any processor may execute PDC_CHASSIS to update the chassis display or read the chassis warnings.

The actual number of chassis displays and chassis warnings that must be implemented, and which processors must execute PDC_CHASSIS at what frequencies is defined by the *Chassis I/O Standard*.

The number of chassis displays, and the mechanism by which they are shared in a multiprocessor system is defined by the *Chassis I/O Standard*.

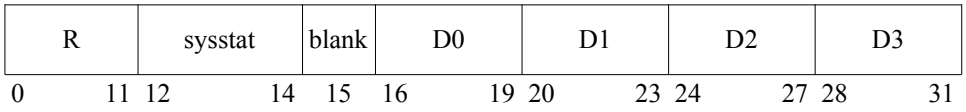
The procedure is best suited to a display with four hex digits, but the display codes are designed to present the maximum error information for each of the display options.

The "**Update chassis display**" option (ARG1=0) displays new data and system state on the chassis display.

The "**Return chassis warnings**" option (ARG1=1) returns warnings pertaining to the fans, batteries, and temperature.

The "**Update display and return warnings**" option (ARG1=2) updates the display and returns the warnings.

The *data* argument specifies the contents of the display, as follows:



The *sysstat* field identifies which of the eight states the system is currently in. The states are defined as follows:

sysstat	System State
000	Off
001	Fault
010	Test
011	Initialize
100	Shutdown
101	Warning
110	Run
111	All On

The values of D0, D1, D2, and D3 are the 4-bit numbers representing the four hex digits on the display. The value generated on the display is the hex representation of these digits, the leftmost digit is D0 and the rightmost is D3.

If the *blank* bit is set, the display should be made blank, if possible, regardless of the values of D0, D1, D2, and D3.

The return parameter *warn* has the following format:

When *r_power* is 0, there is no failure of any redundant chassis component (such as a fan or power supply). A nonzero value of *r_power* identifies a failed redundant chassis component; the encoding of the nonzero values is HVERSION dependent.

The value of *b_low* = 0 implies that the battery is good (or that this feature is not provided). The value of *b_low* = 1 implies that the battery is low, so memory may not be preserved during a power failure.

The value of *t_low* = 1 if the product temperature has exceeded the Temp_Low threshold.

The value of *t_mid* = 1 if the product temperature has exceeded the Temp_Mid threshold.

Processors which do not provide warning detection must return 0 for *warn*.

PDC_CONFIG (index 17)

Purpose: To request deconfiguration and reconfiguration of modules, and to obtain configuration status information.

Options: The implementation of options ARG1=0 and ARG1=1 as a pair is optional. However, if either one is implemented, the other must be implemented as well. Option ARG1=2 is required.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4	ARG5	ARG6	ARG7
Deconfig module	0	R_addr	hpa	R	R	R	R
Reconfig module	1	R_addr	hpa	R	R	R	R
Return config info	2	R_addr	hpa	R	R	R	R

Returns:

Description	RET[0]
Deconfig module	R
Reconfig module	R
Return config info	config_info

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
1	Multiple deconfiguration attempts The requested deconfiguration has already been performed, or an earlier request for deconfiguration was accepted. Returned only by option ARG1=0. REQUIRED.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-4	Absent module There is no module present at the specified hpa. REQUIRED.
-5	Last processor/Insufficient memory The request for deconfiguration was denied because the module for deconfiguration is the last of its type or the remaining amount of memory would have been less than 256 Kbytes. Returned only by option ARG1=0. REQUIRED.
-6	Cannot provide functionality The requested functionality cannot be provided for the specified hpa. REQUIRED. Applicable only for options ARG1=0 or ARG1=1.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: The PDC_CONFIG procedure is used to request deconfiguration and reconfiguration of modules on the central bus. This procedure is also used to obtain the configuration status of modules on the central bus.

The deconfiguration or reconfiguration is performed at the next hard boot. Module deconfiguration and reconfiguration must not be performed during soft boot.

PDC must maintain the module configuration status uniquely across the system as tertiary state, and preserve this information across PDCE_RESET and PDCE_TOC.

SUPPORT NOTE

If deconfiguration and reconfiguration is considered to be absolutely necessary, a soft boot may be followed by a hard boot (after a dump) to perform deconfiguration and reconfiguration.

The "**Deconfig module**" option (ARG1=0) is used to request the deconfiguration of a faulty module on the central bus. PDC must deconfigure the module corresponding to the *hpa* argument at the time of the next hard boot. This call overrides any previous "Reconfig module" option call to the same hpa.

The last processor and memory modules in the system must not be deconfigured. PDC must deny a request to deconfigure the last processor or memory module, and must return a status of -5. Additionally, to improve availability, PDC must ensure that the system has at least 256 Kbytes after deconfiguration.

This call must have no effect on the target module's state. However, the actual deconfiguration during the next hard boot may change the register set state. A deconfigured module may optionally choose not to assert PATH_SLAVE_ACK to all transactions. In such a case, the module must return a status of -4 for the PDC_IODC "Get entry point" option. Also, PDC_ADD_VALID must return a status of 1 or 2 in this case. Alternatively, if the module asserts PATH_SLAVE_ACK to any transaction, it must allow the PDC_IODC "Get entry point" option to complete normally.

PROGRAMMING NOTE

Since PDC_IODC may not return status -4 for a deconfigured module, PDC_CONFIG (ARG1 = 2) should also be called to determine the system configuration.

If this call is issued to any memory module belonging to an interleaved group, the entire interleaved group must be deconfigured.

A deconfigured module must not request bus operations. Further, deconfigured processors must not participate in cache coherence protocols or monarch selection.

SUPPORT NOTE

It is suggested that the operating system inform the user through the system console that deconfiguration will take place at the next hard boot, after a deconfiguration request has been accepted.

The "**Reconfig module**" option (ARG1=1) is used to request the reconfiguration of a module on the central bus specified by the *hpa* argument. The *hpa* argument for this call must represent the hpa that the module would respond to when it is configured. This call overrides any previous "Deconfig module" option call to the same hpa.

If this call is issued to any memory module belonging to an interleaved group, the entire interleaved group must be reconfigured.

The "**Return config info**" option (ARG1=2) is used to obtain the configuration status of the modules on the central bus. The legal values for *config_info* are as follows:

config_info	Configuration Status
0	Module configured and not slated to be deconfigured
1	Module deconfigured and not slated to be reconfigured
2	Module configured, but slated to be deconfigured
3	Module deconfigured, but slated to be reconfigured

The *hpa* argument for this call must represent the hpa that the module would respond to when it is configured or the hpa if configured.

PDC_COPROC (index 7)

Purpose: To identify the coprocessors attached to the processor.

Arguments:

Description	ARG1	ARG2
Return coprocessor configuration	0	R_addr

Returns:

Description	RET[0]	RET[1]
Return coprocessor configuration	ccr_functional	ccr_present

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
1	Error detected, partial use possible This status must be returned if *ccr_functional* does not equal *ccr_present*. CONDITIONAL. Must be used if coprocessors are tested during the call.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: The "**Return coprocessor configuration**" option (ARG1=0) returns parameters which describe the presence and status of the coprocessors attached to the processor. Following PDC_COPROC, the state of all coprocessors is HVERSION dependent.

ENGINEERING NOTE

The coprocessors may be optionally tested during the call. It is recommended that the test be limited to a simple GO/NO GO test.

---

ccr_functional specifies which coprocessors are present and functional. A set bit indicates that the corresponding coprocessor is both present and, if tested, has passed the test. The format of *ccr_functional* is the same as the CCR (CR10). If *ccr_functional* is moved into the CCR, each present and functional coprocessor is enabled.

ccr_present specifies which coprocessors are present. A set bit indicates that the corresponding coprocessor is present. If the bit is set in *ccr_present* and cleared in *ccr_functional*, the coprocessor is present but has failed a functional test. The format of *ccr_present* is the same as the CCR (CR10).

The operating system must call PDC_COPROC during boot and powerfail recovery to determine which coprocessors are present and functional. A state restore sequence of a valid coprocessor state must be used after the PDC_COPROC call to enable use by the OS.

If the coprocessors require initialization upon power on, and PDCE_RESET does not perform that initialization, then the coprocessors must be initialized by PDC_COPROC.

PDC_DEBUG (index 14)

Purpose: To return the hard physical address of the processor and to indicate which modules on the bus exist on the same board as the processor.

Options: Option ARG1=1 is required for processors on a bus which implements On Line Replacement (OLR). Option ARG1=1 is HVERSION dependent for other processors.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4	ARG5	ARG6	ARG7
Return processor HPA	0	R_addr	HV	---	---	---	---
Return modules	1	R_addr	R	R	R	R	R

Returns:

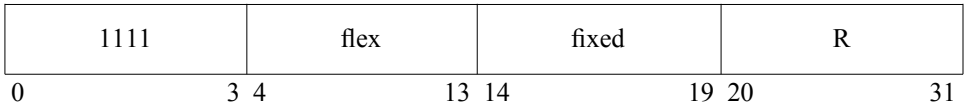
Description	RET[0]	RET[1]
Return processor HPA	hpa	R
Return modules	mods_0	mods_1

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: The "**Return processor HPA**" option (ARG1=0) returns the hard physical address of the processor.

The return parameter *hpa* contains the processor's HPA, in the following format:



The "**Return modules**" option (ARG1=1) returns two 32-bit bitmasks indicating which modules on the bus exist on the same board as the processor. A set bit indicates that the module is on the same board while a clear bit indicates it is not. Bits 0-31 in *mods_0* correspond to modules 0-31. Bits 0-31 in *mods_1* correspond to modules 32-63. If this option is not implemented, software must assume that all modules on the bus are implemented on the processor board.

PDC_IODC (index 8)

Purpose: To obtain the I/O-dependent code for a module and to emulate architected functionality in processor-dependent memory modules.

Options: Options ARG1=2 and ARG1=4 are defined for use with processor-dependent memory modules. Processors that do not support processor-dependent memory do not provide these options. Option ARG1=5 is defined for use with interleaved processor-dependent memory. Processors that do not support interleaved memory do not provide this option.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4	ARG5	ARG6
Get entry point	0	R_addr	hpa	index	memaddr	count
Nondestructive init	2	R_addr	hpa	spa	---	---
Return and clear errors	4	R_addr	hpa	spa	---	---
Identify primary	5	R_addr	hpa	---	---	---

Returns:

Description	RET[0]	RET[1]	RET[2]	RET[3]
Get entry point	actcnt	R	R	R
Nondestructive init	stat	max_spa	max_mem	R
Return and clear errors	stat	resp	info	req
Identify primary	primary_hpa	R	R	R

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
2	Unrecoverable memory module error (some memory usable) A memory error was detected during initialization but some memory is still usable. Returned only by option ARG1=2. CONDITIONAL. Must be used if memory is partially configurable after an error.
1	Recoverable memory module error The call completed normally and the returned results are valid. The procedure encountered an error which it was able to correct completely. Returned only by option ARG1=2. CONDITIONAL. Must be used if the implementation performs error recovery.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-4	Operation completed with requestor bus error The processor detected a requestor bus error during the READ operation to the IO_DC_DATA register of the target HPA (ARG3). The caller may assume that no IODC exists at the target HPA. REQUIRED. Returned only by option ARG1 = 0.
-5	Valid IODC for this module, but invalid index specified in ARG4 REQUIRED. Returned only by option ARG1=0.
-6	Requested IODC exceeds *count* bytes REQUIRED. Returned only by option ARG1=0.

-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.
-18	IODC checksum error REQUIRED. Returned only by option ARG1=0.
-19	Satellite not associated with any primary The target HPA (ARG3) is a satellite which is not configured to be part of any interleave group and thus is not associated with any primary. For example, a faulty satellite might not be configured. REQUIRED. Returned only by option ARG1=5.

Description: The "**Get entry point**" option (ARG1=0) provides access to the IODC of the module specified by the *hpa* argument. Software is required to use the option for all IODC access, rather than read the IODC directly. The caller allocates a buffer in memory into which the option deposits the requested IODC. The buffer is at memory location *memaddr* and is *count* bytes in length. The value of *memaddr* must be word aligned; *count* is an unsigned 32-bit integer and must be a multiple of four. The entry point to get is selected by *index*, which is an unsigned integer in the range [0..255]. The return parameter *actcnt* is the actual number of bytes in the entry point selected. *actcnt* is an unsigned 32-bit integer and must be a multiple of four.

The IODC data bytes are returned in the following format in the *memaddr* buffer:

memaddr

byte 0	byte 1	byte 2	byte 3
byte actcnt-4	byte actcnt-3	byte actcnt-2	byte actcnt-1

The "Get entry point" option treats *index*=0 as a special case. First, *index*=0 does not return IODC entry point 0, but instead returns the first 16 bytes of the module's IODC. Second, RET[0] is HVERSION dependent because modules can provide three different subsets of the first 16 IODC bytes. It is the responsibility of the caller to determine from byte 3 of the *memaddr* buffer (it is the IODC_TYPE byte) which (if any) of the other bytes are valid. The caller must provide 16 bytes of storage, beginning at *memaddr*. The argument *count* is HVERSION dependent.

The existence of a module at the target HPA (ARG3) must be verified by PDC_IODC before any transfer of data is attempted. The caller is not required to have called PDC_ADD_VALID before calling PDC_IODC. PDC_IODC must return status value -4 if it detects a requestor bus error while reading the IO_DC_DATA register of the target HPA. It must also return status value -4 when deconfigured if it does not normally assert PATH_SLAVE_ACK in this state.

When verifying the HPA of the target module, PDC_IODC must conform to the requirements set forth for PDC_ADD_VALID. These requirements include the detection of other HPMC conditions besides requestor bus errors, and the clearing of the soft errors in the bus converters on the path to the target HPA. (See the Description section in the PDC_ADD_VALID page).

If a processor has a data or instruction cache, PDC_IODC must flush the entries in the data or instruction cache that correspond to the memory buffer allocated by the caller.

The following PDC_IODC algorithm validates the existance of a module, and accesses a location (the type field) in the IODC. Any IODC access algorithms should begin in this manner.

1. Read the module's IO_DC_DATA register. If a bus error results, the module is presumed not to exist and the sequence is terminated.
2. Write the value 3 to IO_DC_ADDRESS to address the module type specifier (the IODC_TYPE byte).
3. Read IO_DC_DATA to obtain the IODC_TYPE byte.

ENGINEERING NOTE

The PDC_IODC procedure allows fixed configurations to use the processor's PDC to emulate a module's IODC. In particular, for native processors without the IO_DC_ADDRESS and IO_DC_DATA registers, the PDC_IODC procedure must be able to identify the processor and return 0 for its IODC_TYPE byte.

PROGRAMMING NOTE

It is possible that a module that is deconfigured, even if physically present, returns a status of -4 to the PDC_IODC "Get entry point" option. Software should use the PDC_CONFIG "Return config info" call in conjunction with the PDC_IODC "Get entry point" option to ascertain the modules that are physically present.

The "**Nondestructive init**" option (ARG1=2) is used to initialize a memory module and to determine the size of the module. It is expected that the operating system will call this option during powerfail recovery and during boot. The option does not change the contents of the data in the memory module and does not do any lengthy array tests. The option sets the SPA of the memory module identified by *hpa* to the base address given by *spa*. If the procedure returns status value -3 it must disable the memory module's SPA.

The format of the return parameter *stat* is the same as the I/O register IO_STATUS.

The return parameter *max_spa* specifies the size of the memory module's SPA space in bytes. *max_spa* must be a power of two, and must be less than or equal to the value specified by the module's IODC_SPA[shift] field.

The return parameter *max_mem* specifies the amount of implemented memory in bytes. *max_mem* must be strictly greater than ½ of *max_spa*.

Executing this option clears the HVERSION-dependent versions of IO_STATUS[fe], IO_STATUS[he], IO_STATUS[se], and IO_STATUS[estat]. The IO_STATUS[sl] bit is not changed by this option.

The sequence of events performed by the "Nondestructive init" option is as follows:

1. Initialize the memory SPA.
2. Return *max_spa* and *max_mem*.
3. Return error status in the RET[0] parameter. Error bits *he*, *se*, *fe*, *estat* are cleared in the HVERSION-dependent IO_STATUS register after the call.

Following are the calling conventions for the "Nondestructive init" option:

- Software must not call the "Nondestructive init" option for a processor-dependent IMM or for a processor-dependent memory module which is a satellite of the IMM.

- Software may issue CMD_RESET to processor-dependent memory modules before calling the "Nondestructive init" option. Software need not check for the completion of the reset command; PDC_IODC must handle this.
- For every processor-dependent memory module which is a satellite of a module other than the IMM, that satellite must be reset or must have 0 written to its IO_SPA register, before its SPA base can be changed by calling the "Nondestructive init" option for the primary memory module of its interleave group.
- Software must never call the "Nondestructive init" option for a processor-dependent memory module which has IODC_SPA[shift] = 0 (a processor dependent satellite module). Effects of such a call are HVERSION dependent.

The "**Return and clear errors**" option (ARG1=4) checks for memory errors and clears the HVERSION-dependent equivalents of IO_STATUS[sl,estat,se,he] for the processor-dependent memory module specified by the *hpa* argument. Only the status of the first error of the highest severity that has occurred since the routine was previously called is returned. If fe = 0, then all error fields must be cleared. If fe = 1, then sl and estat must not be cleared and se and he are HVERSION dependent.

The formats of *stat*, *resp*, *info*, and *req* are the same as the I/O registers IO_STATUS, IO_ERR_RESP, IO_ERR_INFO, and IO_ERR_REQ, respectively. If the memory module does not support logging of the error requestor, the *req* return parameter is 0.

The "Return and clear errors" option is equivalent to the following sequence for an architected memory module:

```
RET[0] ← IO_STATUS;
RET[1] ← IO_ERR_RESP;
RET[2] ← IO_ERR_INFO;
RET[3] ← IO_ERR_REQ;
IO_COMMAND ← CMD_CLEAR.HE;
```

The "**Identify primary**" option (ARG1=5) identifies the primary memory module of an interleave group. ARG3 must be the hpa of a processor-dependent memory module which has IODC_SPA[shift] = 0. If ARG3 is the hpa of a processor-dependent memory module configured as a satellite in an interleave group, then the hpa of the group primary is returned as *primary_hpa*. If ARG3 is the hpa of a processor-dependent memory module which has IODC_SPA[shift] = 0 but is not configured in an interleave group, then status -19 is returned.

PDC_MEM (index 20)

Purpose: To identify pages in which a memory error has been detected so that the module can be configured at boot time, but the page with the error will not be accessed by software. The firmware table which contains this information will be referred to as the Page Deallocation Table (PDT).

Arguments:

Description	ARG1	ARG2	ARG3	ARG4
Return Information	0	R_addr	R	R
Add Page	1	R_addr	PDT_addr	R
Clear PDT	2	R_addr	R	R
Read PDT	3	R_addr	PDT_entries_ptr	R
Reset Clear Flag	4	R_addr	R	R
Set Good Memory	5	R_addr	good_mem	R

Returns:

Description	RET[0]	RET[1]	RET[2]	RET[3]	RET[4]
Return Information	PDT_size	page_entries	PDT_status	dbe_loc	good_mem
Add Page	R	R	R	R	R
Clear PDT	R	R	R	R	R
Read PDT	page_entries	R	R	R	R
Reset Clear Flag	R	R	R	R	R
Set Good Memory	R	R	R	R	R

Status:

Value	Description
5	Single bit error replaced. A double bit error replaced a single bit error, because the table was full. REQUIRED.
4	Duplicate entry not added. An entry for the page to be added is already in the PDT. REQUIRED.
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-11	Table full. An attempt to add an entry to the PDT failed because the table was full. REQUIRED.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: The PDC_MEM procedure is used to identify memory pages in which errors have been detected so that they will not be accessed by software.

PDC must maintain the PDT and the mapping between the real page number of each memory location and its physical hardware memory address (module, HVERSION dependent portion of a module, and address within portion of a module) uniquely across the system as tertiary state, and

preserve this information across boot and powerfail.

SUPPORT NOTE

If memory modules are physically swapped within a system, the PDT could be in error. This could cause unwanted effects.

Good locations would be erroneously listed, deallocating good memory, the PDT could become full, disabling page deallocation, or recurring but non-persistent uncorrectable errors could cause HPMCs.

Moving memory module should only be used as a last resort, and the PDT should be cleared prior to performing the physical swap.

PROGRAMMING NOTE

There are several conditions which cause entries to added to the PDT.

1. During destructive memory initialization when the system is being booted.
2. When an HPMC occurs during normal operation.
3. When software discovers a pattern or non-persistent errors to the same location.

The "**Return Information**" option (ARG1=0) returns information about the PDT. The *PDT_size* parameter returns the maximum number of entries in the PDT. The *page_entries* parameter returns the number of pages currently listed to be deallocated. *page_entries* is an unsigned 32 bit integer.

The *PDT_status* parameter returns information concerning the action of the memory initialization process during boot. The format of the PDT_status paramater is as follows:

The *cs* or cleared status bit indicates whether the PDT has been cleared since the last call to the "**Reset Clear Flag**" option. A value of 0 indicates that "**Reset Clear Flag**" has been called since the the last time that the PDT was cleared. A value of 1 indicates that the PDT has been cleared since the last time that "**Reset Clear Flag**" was called. The PDT will be cleared whenever the CLEAR PDT option of PDC_MEM is called or whenever memory initialization must clear the PDT because of reinterleaving.

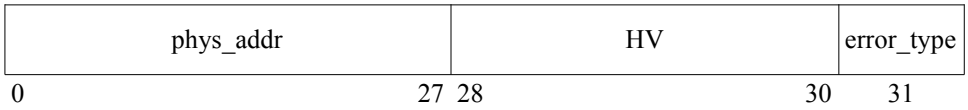
A value of 1 in the *is* or interleaving changed bit indicates that during memory initialization, memory interleaving had to be changed in such a way that may reduce performance in order to enlarge the error free portion of initial memory. A value of 0 in the interleaving changed bit indicates that memory initialization did not reinterleave to enlarge the error free portion of initial memory.

The *dbe_loc* parameter returns the location in memory where the first double bit error was detected. If no errors were detected this parameter is set to the value (2^32)-1 (0xFFFFFFFF). *Dbe_loc* is in byte address format. *dbe_loc* is an unsigned 32 bit integer.

The *good_mem* parameter returns the target value used by memory initialization to provide initial memory without double bit errors. *good_mem* is a 32 bit unsigned integer indicating the number of bytes of good memory required.

The "**Add Page**" option (ARG1=1) adds a page to the PDT so that it will not be referenced by software. The PDT_addr argument indicates the errored location whose page is to be added to the

PDT. The format of PDT_addr is as follows:



The *phys_addr* fields specifies bits 0 through 28 of the real address where the error was detected. The *error_type* field is used to specify the type of error. The value 0 is used to indicate a double bit error. The value 1 is used to indicate a single bit error.

The "**Clear PDT**" option (ARG1=2) removes all entries from the PDT. In addition, it causes subsequent calls to the "**Return Information**" option to return a value of 1 for the cleared status bit of the *PDT_status* parameter until "**Reset Clear Flag**" is called.

The "**Read PDT**" option (ARG1=3) reads the PDT, and return the information in a memory area provided by the caller. The *PDT_entries_ptr* argument is the pointer to that memory area. The memory area must be 4*PDT_size bytes in length and must be double word aligned. The *page_entries* return parameter indicates the number of pages listed to be deallocated. For each entry in the PDT, one word in the format of PDT_addr is returned. The values are returned in a set of contiguous locations beginning at the location referenced by *PDT_entries_ptr*. Locations beyond the last PDT entry are HVERSION dependent.

The "**Reset Clear flag**" option causes subsequent calls to the "**Return Information**" option to return a value of 0 for the cleared status bit of the *PDT_status* parameter until the PDT is cleared, either by a call to the "**Clear PDT**" option, or by PDC selftest during boot.

The "**Set Good Memory**" option sets the value of the *good_mem* parameter returned by the "**Return Information**" option.

PDC_MODEL (index 4)

Purpose: To return the version numbers, identifiers, and capabilities of a processor, to set the BOOT_ID of a processor, to return the version numbers of processor components, to return the system model information of a system, and to enable and disable the execution of product-specific instructions.

Options: Category A processors may optionally provide option ARG1=1; category B processors must provide option ARG1=1.
Processors may optionally provide option ARG1=2 if they wish to support CVERSIONs.
Processors that support any product-specific instructions must provide options ARG1=4 and ARG1=5. Processors that do not support any product-specific instructions must not provide options ARG1=4 and ARG1=5.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4	ARG5	ARG6	ARG7
Return info	0	R_addr	HV	---	---	---	---
Set BOOT_ID	1	BOOT_ID	HV	---	---	---	---
Return versions	2	R_addr	c_index	---	---	---	---
Return system model	3	R_addr	OS_ID	mod_addr	R	R	R
Enable specific	4	R_addr	enable_key	R	R	R	R
Disable specific	5	R_addr	disable_key	R	R	R	R
Return CPU ID	6	R_addr	R	R	R	R	R
Return capabilities	7	R_addr	R	R	R	R	R
Return boot test options	8	R_addr	R	R	R	R	R
Set boot test options	9	R_addr	tests_off	tests_on	R	R	R

Returns:

Description	RET[0]	RET[1]	RET[2]	RET[3]	RET[4]
Return info	HVERSION	SVERSION	HV	BOOT_ID	SW_ID
Set BOOT_ID	---	---	---	---	---
Return versions	CVERSION	R	R	R	R
Return system model	mod_len	R	R	R	R
Enable specific	R	R	R	R	R
Disable specific	R	R	R	R	R
Return CPU ID	CPU_ID	R	R	R	R
Return capabilities	caps	R	R	R	R
Return boot test options	current_tests	tests_supported	default_tests	R	R
Set boot test options	R	R	R	R	R

Description	RET[5]	RET[6]	RET[7]	RET[8]	
Return info	SW_CAP	arch_rev	---	---	
Set BOOT_ID	---	---	---	---	
Return versions	R	R	R	R	
Return system model	R	R	R	R	
Enable specific	R	R	R	R	
Disable specific	R	R	R	R	
Return CPU ID	R	R	R	R	
Return capabilities	R	R	R	R	
Return boot test options	R	R	R	R	R
Set boot test options	R	R	R	R	R

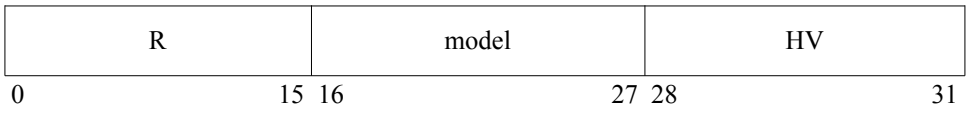
Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.

1	Valid *c_index* but no CVERSION returned Returned only by option ARG1=2. CONDITIONAL. Must be used if the implementation does not have a CVERSION for each valid *c_index*.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-4	Invalid *c_index* (nonexistent component) Returned only by option ARG1=2. REQUIRED.
-5	Invalid *OS_ID* (unknown operating system) Returned only by option ARG1=3. REQUIRED.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.
-20	Unsupported key A key not supported by this processor was requested. Returned only by options ARG1=4 and ARG1=5. REQUIRED.

Description: The "**Return info**" option (ARG1=0) returns the version numbers, identifiers, and capabilities of the processor module.

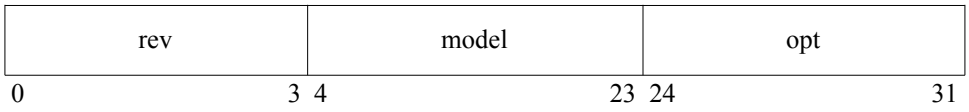
The value of **HVERSION** specifies the hardware version number for the processor as follows:



The *model* field specifies the hardware implementation, and is changed as required for implementations with incompatible diagnostic and/or system dependent software functionality.

The five most significant bits of *model* are the bus ID of the bus on which the processor module is located. See Section 5.5.6.1, IODC Data Bytes, for assignment of identifiers.

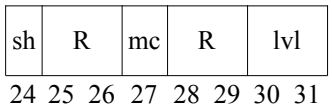
The value of **SVERSION** specifies the software version number for the processor as follows:



The *rev* field is X'0 for all native processors.

The *model* field is X'00004 for all native processors.

The definition of the SVERSION[opt] byte for native processors is:

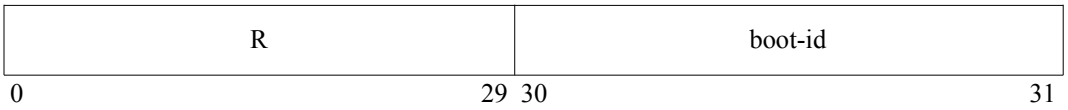


sh Indicates if shadow registers are present. If shadow registers are implemented, this field is a 1; this field is a 0 otherwise.

mc Specifies the module category. This bit is 0 for category A processors and 1 for category B processors.

lvl Specifies the native processor capability level (0, 1, 2, and 3 for Level_0, Level_1, Level_1.5, and Level_2, respectively).

The **BOOT_ID** (boot identifier) word is used during monarch selection (see the PDCE_RESET description in Section 3.2, PDC Entry Points). BOOT_ID has the following format:



For a category A processor that does not provide the ARG1=1 option, RET[3] is HVERSION dependent.

For category B processors, BOOT_ID must be set to 2 at manufacture time.

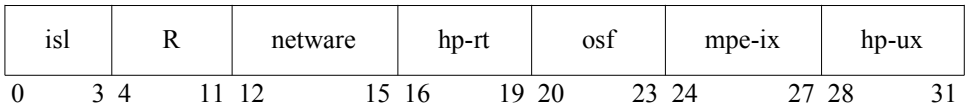
The **SW_ID** (software identifier) word is a system-wide resource, and is a unique 32 bit unsigned integer (each system shipped has a different value). The SW_ID is used for software licensing and security. The SW_ID of each system must be set to a unique value at manufacture.

If the FRU containing the SW_ID is replaced in the field, a secure mechanism must be used to set the SW_ID in the new FRU equal to the SW_ID in the original FRU. The mechanism is HVERSION-dependent for category A processors and SVERSION-dependent for category B processors. This ensures that the SW_ID of the system is unchanged and that software licensed to the system will still execute.

SUPPORT NOTE

The method by which SW_IDs are allocated is not architected, but FES, DSTL, and SAL have developed a standard algorithm for calculating the SW_ID. The algorithm is based on the serial number, product number, and manufacturing site of the system. Diagnostic tools exist to translate between these three parameters and the SW_ID, and vice versa. Some series 9000/700 machines use LAN_ID instead of serial number.

The **SW_CAP** (software capabilities) word specifies the operating system capabilities of the processor module as follows:



SW_CAP consists of the *netware* field for Novell Netware, the *hp-rt* field for HP-RT, the *osf* for OSF/1, the *mpe-ix* field for MPE-iX, and the *hp-ux* field for HP-UX. The operating system capabilities associated with each of these fields are as follows:

isl	Description
0	No ISL capability restrictions in effect
1-15	Defined by ISL

netware	Description
0	No Novell Netware capability restrictions in effect
1-15	Defined by the Novell Netware operating system

hp-rt	Description
0	No HP-RT capability restrictions in effect
1-15	Defined by the HP-RT operating system

osf	Description
0	No OSF capability restrictions in effect
1-15	Defined by the OSF operating system

mpe-ix	Description
0	No MPE-iX capability restrictions in effect
1-15	Defined by the MPE-iX operating system

hp-ux	Description
0	No HP-UX capability restrictions in effect
1-15	Defined by the HP-UX operating system

SW_CAP is a per-processor resource. When each processor module is manufactured, the associated SW_CAP is set to an appropriate value. When a processor module is replaced, the SW_CAP of the new processor module must be set equal to the SW_CAP of the processor module being replaced. SW_CAP is updated when the operating system capabilities of the processor module are changed (e.g., when a system is upgraded).

Each implementation must provide an HVERSION-dependent mechanism, accessible by software, to change the SW_CAP value. The ability to access the mechanism to change this value must be restricted to software executing at privilege level 0.

ENGINEERING NOTE

The architecture makes no requirement as to where the SW_CAP value is physically stored; however, the intended use of SW_CAP is dependent on implementations making SW_CAP physically part of the processor module. SW_CAP is envisioned as residing on the processor module when the processor module is installed in a system as an upgrade that replaces another processor, or is installed in a system as an additional processor.

SUPPORT NOTE

While the architecture allows each processor in a system to have a different value of SW_CAP, the field support organization does not have a vehicle to manage multiple values of SW_CAP for a single system. There are no plans to have more than one value of SW_CAP for a system.

The *arch_rev* field specifies the revision of the architected instruction set supported by the processor. The *arch_rev* values are assigned by Computer Systems Architecture. Values of *arch_rev* > 0 correspond to architected supersets of the original instruction set, as shown below:

arch_rev	Description
0	PA-RISC 1.0
4	PA-RISC 1.1

The *potential_key* word indicates which unprivileged, product-specific instructions are supported by the processor. Each bit position in *potential_key* corresponds to a distinct group of product-specific instructions. If a bit in *potential_key* is set to 1, then the processor supports the corresponding group of product-specific instructions; otherwise, it does not support them. Thus, *potential_key*=0 indicates a processor that supports no product-specific instructions at all. The *key* values are assigned by the System Architecture Lab. A table showing the correspondence between bit positions and groups of product-specific instructions follows:

key	Description
X'00000001	Opcode X'0F does quadword stores (as in PCX-S)
X'00000002	FSTWS, FSTWX, FLDWS, FLDWX generate word transactions to I/O space (as in PCX-T)
X'00000004	Floating point includes reciprocal square root (as in PCX-S)
X'00000008	Flush data cache includes graphics flushes (as in PCX-S & PCX-T)
X'00000010	Multimedia halfword shift-and-adds. (as in PCX-L and PCX-U)
X'00000020	Multimedia halfword adds and subtracts. (as in PCX-L and PCX-U)
X'00000040	Byte swapping stores (as in PCX-L)
X'00000080	Speculative Data Prefetch on Loads of GR0

The *current_key* word indicates which product-specific instructions are enabled for execution on the processor. The allowable values of *current_key* are 0, which indicates that the product-specific instructions are disabled (or that the processor supports no product-specific instructions at all), and *potential_key*, which indicates that the product-specific instructions are enabled.

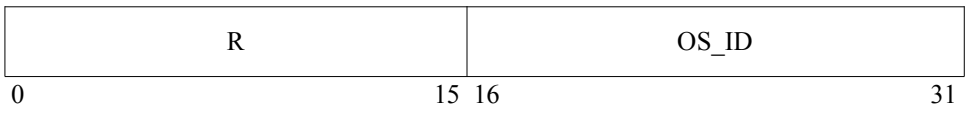
The "**Set BOOT_ID**" option (ARG1=1) is used to set the BOOT_ID of the processor module.

The "**Return versions**" option (ARG1=2) is used to identify the version number of each processor component. The argument *c_index* is the processor component index; *c_index* is an unsigned 32-bit integer. The meaning of each index is HVERSION dependent, but indexes must be assigned sequentially beginning at 0. This allows software to identify all components by repeated calls to PDC_MODEL with increasing values of *c_index* until status -4 (Invalid *c_index*) is returned.

The **CVERSION** (component version) word specifies the version number for a component associated with the processor. The interpretation of CVERSION depends on the HVERSION of the processor.

The "**Return system model**" option (ARG1=3) returns the system model information of the system.

The *OS_ID* argument is used to identify the operating system, and has the following format:



Refer to the description of the PDC_STABLE procedure for further information about *OS_ID*.

The *mod_addr* argument points to a byte-aligned array into which the system model of the operating system indicated by *OS_ID* is returned. A system model string can be up to 80 characters long and its length is returned in the 32 bit unsigned integer *mod_len*. The format and content of each string is defined by the operating system to which it applies.

PROGRAMMING NOTE

Although "Return system model" is a required option, some old processors do not provide it. Callers should be prepared to generate system model information for such processors, perhaps via a lookup table indexed by HVERSION.

ENGINEERING NOTE

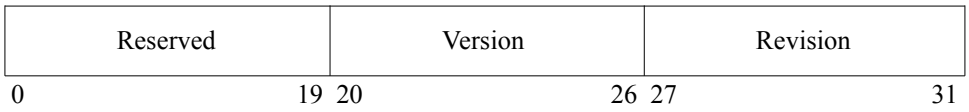
Implementations must choose the correct system model information to return when the same processor module and PDC are used in seperately marketed products.

Implementations are encouraged to provide an HVERSION-dependent mechanism, accessible by software, to change the system model information.

The "**Enable specific**" option (ARG1=4) enables the product-specific instructions indicated by *enable_key*. The only allowable value for *enable_key* is the same bit mask as returned in *potential_key* by the "Return info" option, otherwise the call returns -20 and all product-specific instructions are disabled.

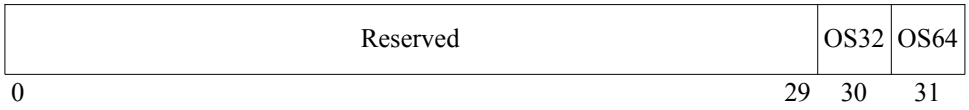
The "**Disable specific**" option (ARG1=5) disables the product-specific instructions indicated by *disable_key*. The only allowable value for *disable_key* is the same bit mask as returned in *potential_key* by the "Return info" option, otherwise the call returns -20 and all product-specific instructions are disabled.

The "**Return CPU ID**" option (ARG1=6) returns a value in *CPU_ID* which uniquely identifies the CPU portion of the processor module. The *CPU_ID* is returned in the following format:



CPU_ID values are defined in Appendix B, Version and Identification Numbers.

The "**Return capabilities**" option (ARG1=7) returns a value in *caps* which denotes the platforms capabilities for supporting 32-bit or 64-bit OSes. The *caps* parametr is returned in the following format:

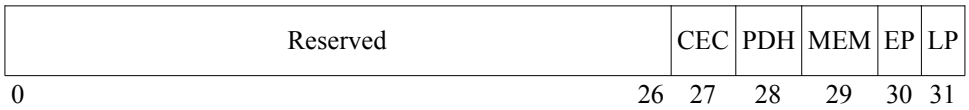


If the OS32 bit is set, the platform is capable of supporting 32-bit OSes. If it is clear, the platform is not capable of supporting 32-bit OSes.

If the OS64 bit is set, the platform is capable of supporting 64-bit OSes. If it is clear, the platform is not capable of supporting 64-bit OSes.

If the call returns "Invalid Argument" for ARG1, indicating the "**Return capabilities** option is not implemented, software should assume that the platform is capable of supporting 32-bit OSes, but is not capable of supporting 64-bit OSes.

The "**Return boot test options**" option (ARG1=8) returns a three values which describe the boot tests suppoted on the platforms, and their current status regarding whether to be run or not. Each return parameter is formatted a boot Test Option Map, shown in the following figure:



- The *CEC* bit refers to Central Electronic Complex tests. This may include I/O bridges and system interconnect media.
- The *PDH* bit refers to Processor Dependent Hardware tests. This may include boot ROM and RAM and any special hardware (eg: semaphores) required by PDC.
- The *MEM* bit refers to Destructive Memory tests. Ths is the set of architected memory tests described in Chapter 11 of this document.
- The *EP* bit refers to Early Processor tests that are run before memory and most of the interconnect and I/O system are available.
- The *LP* bit refers to Late Processor tests that are run after memory, interconnect, and I/O are available.

The *current_tests* return value indicates which tests are currently set to be run on each boot of the system.

The *tests_supported* return value indicates which tests could be run on the system.

The *default_tests* return value indicates which tests are enabled when the system is manufactured and shipped.

In each case, a 1 bit indicates enabled, and a 0 bit indicates disabled.

The "**Set boot test options**" option (ARG1=9) provide a mechanism to enable and disable specific tests. The *tests_off* argument selects a specific set of tests to disable. The *tests_off* argument is formatted as a Boot Test Option Map, and a 1 bit in a specific position indicates to disable the corresponding test. The *tests_on* argument selects a specific set of tests to enable. The *tests_on* argument is formatted as a Boot Test Option Map, and a 1 bit in a specific position indicates to enable the corresponding test.

If an attempt is made to enable and disable the same test at the same time, or if an attempt is made to enable or disable an unsupported test, or if a bit is set in a reserved field, "**Set boot test options**" will return an Invalid Argument Status. If a test which is already disable is disabled, or a test which is already enabled is enabled, "**Set boot test options**" will simply perform the action normally.

PDC_NVOLATILE (index 11)

Purpose: To provide access to Non-Volatile Memory.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4
Read data	0	nvaddr	memaddr	count
Write data	1	nvaddr	memaddr	count
Return size	2	R_addr	HV	---
Verify contents	3	HV	HV	---
Initialize	4	HV	HV	---

Returns:

Description	RET[0]
Read data	---
Write data	---
Return size	size
Verify contents	---
Initialize	---

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
1	Correctable error The call completed normally and the returned results are valid. The procedure encountered an error which it was able to correct completely. Returned only by options ARG1=0, 1, 3, and 4. CONDITIONAL. Must be used if PDC_NVOLATILE performs error recovery.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-5	Invalid NVM contents Returned only by options ARG1=0, 1, 3, and 4. REQUIRED.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. CONDITIONAL. Must be returned by ARG1=0 or 1 if *nvaddr+count > size*. Otherwise, the procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: **Non-Volatile Memory** (NVM) is an optional storage area used to maintain system parameters during power outages. NVM is required to retain its contents even if the card containing it is removed from the backplane. NVM is used during boot and in the event of an OS panic.

Non-Volatile Memory is a system-wide resource shared by all processors in a multiprocessor system. Software must ensure that at most one call to PDC_NVOLATILE is in progress at any one time.

The integrity of NVM is verified in an HVERSION-dependent fashion.

Non-Volatile Memory must not have a lifetime write cycle limit.

The "**Read data**" option (ARG1=0) transfers *count* bytes from NVM address *nvaddr* to memory address *memaddr*. *count* is an unsigned 32-bit integer which is a multiple of four. *nvaddr* and *memaddr* must be word aligned. The call must return -10 if *nvaddr+count* > *size*.

Reads must validate data integrity for the bytes being read. If a checksum algorithm is used, this may involve validating all of NVM. If the data integrity check fails, the call must return -5 and (if possible) the potentially bad data.

The "**Write data**" option (ARG1=1) transfers *count* bytes from memory address *memaddr* to NVM address *nvaddr*. *count* is an unsigned 32-bit integer which is a multiple of four. *nvaddr* and *memaddr* must be word aligned. The call must return -10 if *nvaddr+count* > *size*.

If a write to NVM is interrupted due to a powerfail, reset or TOC, and if PDC cannot guarantee that the write completed, then it must ensure that data integrity checks fail on subsequent accesses to NVM.

ENGINEERING NOTE

A recommended method to check the validity of NVM is to include a checksum in a non-architected tertiary state storage area. This checksum should be designed such that NVM filled with all zeroes or all ones will not generate a valid checksum.

If a write to NVM is interrupted by a powerfail, reset, or TOC, hardware implementations should attempt to limit the extent of damage to the words that were being modified. This rule applies except for genuine hardware failures or sudden power failures. For such failures, the extent of damage cannot be predicted.

The "**Return size**" option (ARG1=2) returns the number of bytes in the processor's NVM. *size* is the number of contiguous bytes implemented in NVM starting from *nvaddr*=0. *size* is an unsigned 32-bit integer and must be a multiple of four.

The "**Verify contents**" option (ARG1=3) verifies that the NVM contents are valid.

The "**Initialize**" option (ARG1=4) sets the entire NVM contents to zero and initializes the validity indicator.

PROGRAMMING NOTE

An algorithm to restore NVM in the event of a failure follows:

1. Copy out the entire contents of NVM into memory.
2. Fix all the bad values by correcting the copied values in memory.
3. Call the "Initialize" option to zero NVM.
4. Write the good copy from memory to NVM.
5. Call the "Verify contents" option to to check for successful completion.

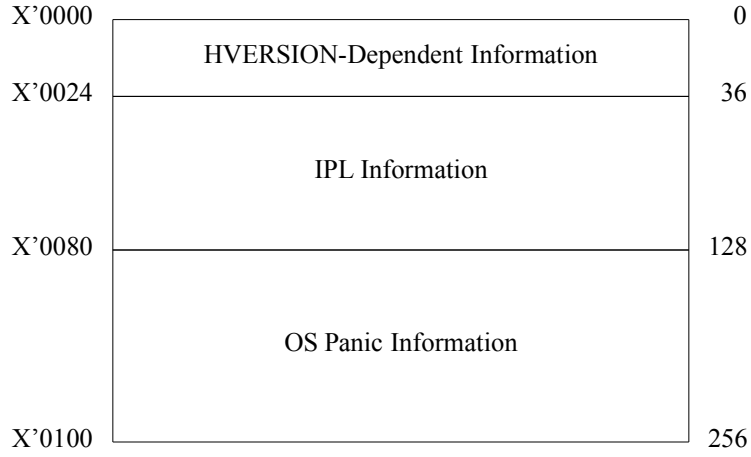
To protect against a powerfail after the "Initialize" option has zeroed NVM, but before the new values are written back, software should back up the contents on disk before zeroing, or be able to reconstruct NVM from the all zero state.

ENGINEERING NOTE

Specific products may choose to implement multiple copies of Non-Volatile Memory to increase the fault tolerance of the system, but this must be transparent to the callers of PDC_NVOLATILE.

Data Format of Non-Volatile Memory

The format of Non-Volatile Memory is as follows:



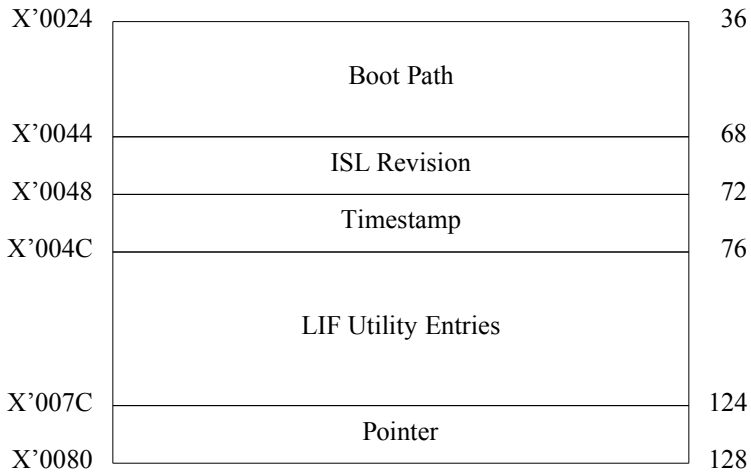
Providing Non-Volatile Memory is optional. If Non-Volatile Memory is provided, it must be at least 256 bytes in size and must be used for the architected purposes.

HVERSION-Dependent Information

The first 36 bytes of NVM are HVERSION dependent.

IPL Information

The 92-byte area starting at address 36 is allocated for IPL to save initialization information. This area is used by ISL to store the following information:



OS Panic Information

The 128-byte area starting at address 128 is allocated for the OS to save panic information. The format of this area is OS_ID dependent.

PDC_PIM (index 3)

Purpose: To access Processor Internal Memory (PIM).

Options: All processors must provide the ARG1=0, ARG1=1, and ARG1=4 options.

Processors that support LPMCs must provide the ARG1=2 option; other processors do not provide the option.

The implementation of the ARG1=3 option is required for category B processors; category A processors do not provide the option.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4
Transfer HPMC data	0	R_addr	memaddr	count
Return size	1	R_addr	HV	---
Transfer LPMC data	2	R_addr	memaddr	count
Transfer Soft boot data	3	R_addr	memaddr	count
Transfer TOC data	4	R_addr	memaddr	count

Returns:

Description	RET[0]	RET[1]
Transfer HPMC data	actcnt	HV
Return size	size	HV
Transfer LPMC data	actcnt	HV
Transfer Soft boot data	actcnt	HV
Transfer TOC data	actcnt	HV

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-5	Invalid PIM contents Returned only by options ARG1=0, 2, 3, and 4. REQUIRED. The event did not occur or the contents are invalid due to power-on, a hard reset, the overwriting rules, or a subsequent call to transfer the same PIM data.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: PIM is a per-processor, HVERSION-dependent storage area in the processor set at the time of an HPMC, LPMC, TOC, or Soft boot. Processors may optionally store PIM information in any of the processor-dependent areas below MEM_FREE.

The purpose of PIM is threefold:

- To help identify a failed FRU for support (for HPMCs and LPMCs)
- To save the processor's state at the time of a TOC or Soft boot for later analysis and, in the case of TOC, to determine if the interrupted process can be resumed.

- To determine what recovery action should be taken.

The storage for any two or more of the four events may optionally be shared. When the storage is shared, the following table specifies the overwriting rules.

		Current PIM Contents			
		HPMC	TOC	Soft Boot	LPMC
Event	HPMC	No	Yes	Yes	Yes
	TOC	No	No	No	Yes
	Soft Boot	No	No	No	Yes
	LPMC	No	No	No	No

If the PIM contents are overwritten or they could not be stored because of the overwriting rules, a PDC_PIM call to transfer the overwritten data must return status -5 to indicate that the PIM contents are invalid.

At power-on or after hard reset, the PIM contents must become invalid.

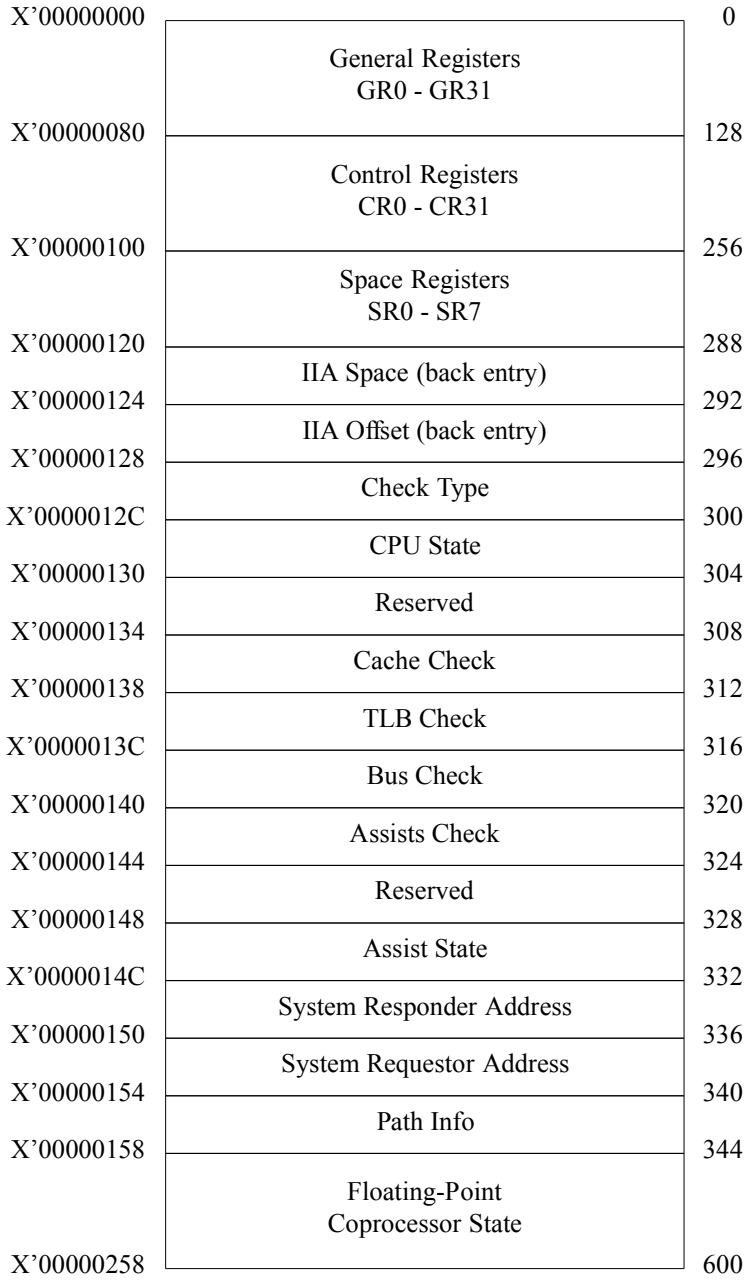
PDC_PIM transfers the contents of PIM into memory. The image of PIM in memory consists of two types of information: information about the processor state, and information about the error. Additionally, some HVERSION-dependent information may be returned.

For an HPMC, information about the processor state and the machine check is returned. For an LPMC, only information about the machine check is returned. For Soft boot, only information about the processor state is returned. For TOC, information about the processor state and the damage done by the TOC is returned.

PDC_PIM uses separate options for transferring HPMC, LPMC, Soft boot, and TOC data. For these options, PDC_PIM transfers data from PIM into the memory address specified by *memaddr*, which is a word-aligned memory address. The value of *count* (which is an unsigned 32-bit multiple of 4) specifies the number of bytes available in the buffer at *memaddr*. One return parameter is defined. The value of *actcnt*, also an unsigned 32-bit multiple of 4, is the actual number of bytes transferred. If the number of bytes in the PIM image exceeds the *count* parameter, only *count* bytes are transferred. Subsequent transfer requests to return the same PIM data must return a status value of -5.

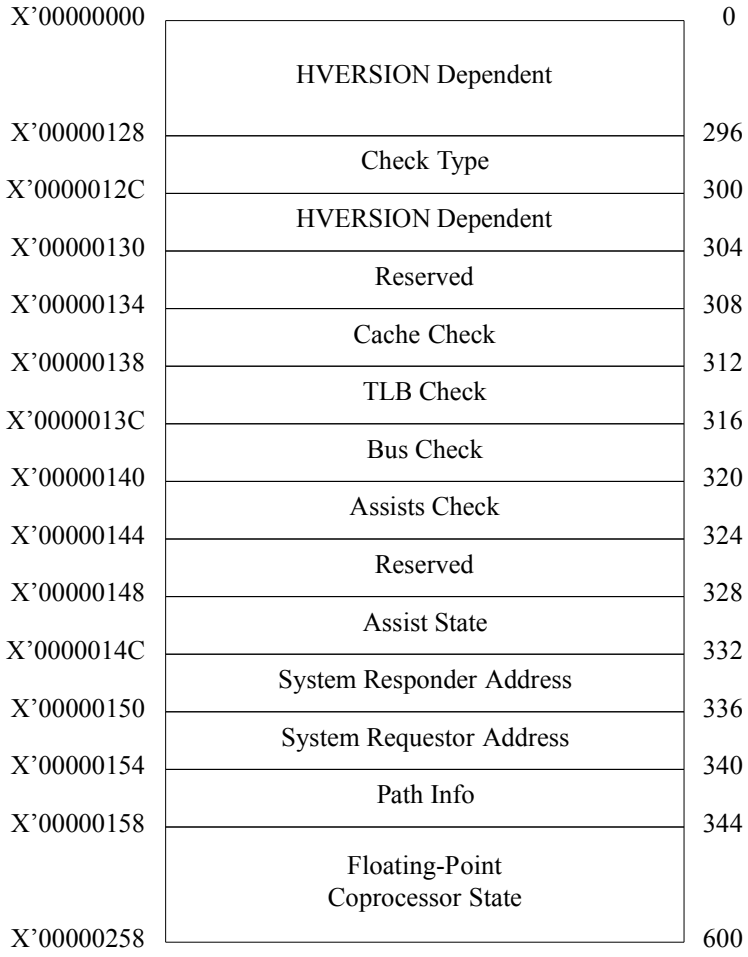
Another option ("Return size") is used to inform the caller how large a buffer is needed to hold the PIM data. The parameter returned by this option applies to all the other options. The option returns *size*, the total size of the PIM image, which is an unsigned 32-bit multiple of 4.

The "**Transfer HPMC data**" option (ARG1=0) returns information about the processor state and the machine check condition in the following format:

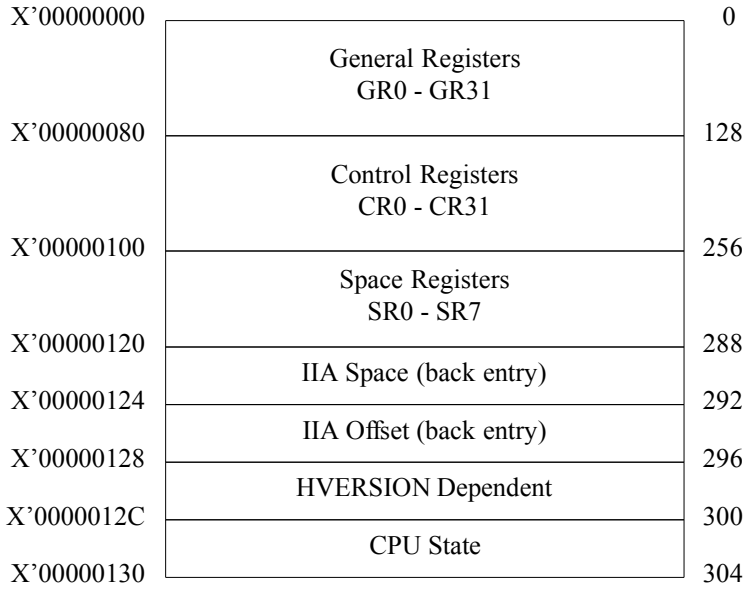


The "**Return size**" option (ARG1=1) returns *size*, the maximum total size of PIM. This option must not modify the contents of PIM or re-enable error collection. The *size* parameter has a constant value for any particular implementation; the operating system can read this once and allocate sufficient buffer space for reading PIM.

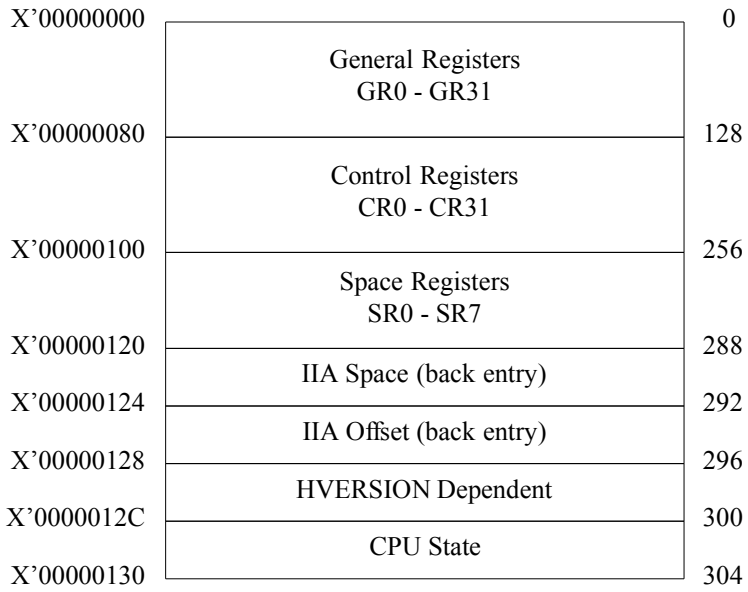
The "**Transfer LPMC data**" option (ARG1=2) returns information about the machine check condition in the following format:



The "**Transfer Soft boot data**" option (ARG1=3) returns information about the processor state in the following format:



The "**Transfer TOC data**" option (ARG1=4) returns information about the processor state and the damage done by the TOC in the following format:



Data Format of PIM

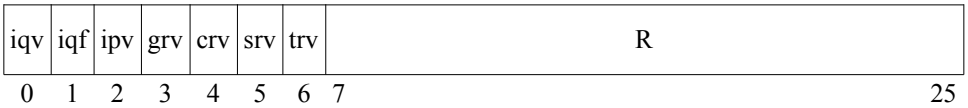
Although different amounts of information are returned for the four types of events, the description of what the various indicators mean is largely independent of which event was logged.

Processor State

A portion of PIM is used to store the processor's General Registers, Control Registers, and Space Registers. The front entries of the IIA Space and IIA Offset Queues (CR17 and CR18) are saved in the Control Registers area, while the back entries of the IIA Space and IIA Offset Queues are saved in the IIA Space and IIA Offset areas, respectively. Storing of the CPU State word is required for HPMCs, TOCs, and Soft Boots, and is HVERSION dependent for LPMCs.

For HPMCs, this processor state reflects the CPU state saved at the time the HPMC interruption was taken and does not necessarily correspond to the time of occurrence of the condition which caused the HPMC.

Several bits in the CPU State word indicate the success of the CPU state save. The format of the CPU State word for these bits is as follows:



Field	Description
iqv	IIA queue Valid. When set, the IIA queue entries are valid. The validity of *iqv* is independent of the value of the IPSW (CR22) Q-bit.
iqf	IIA queue Failure. The front element of the IIA queue points at the instruction that caused the failure. This indicates that the current instruction stream is synchronized with the failure. This bit is only defined for HPMCs; it is HVERSION dependent for TOCs and soft boots.
ipv	IPRs Valid. When set, the IIR (CR19) contains the instruction causing the failure. The IOR (CR21) and ISR (CR20) are HVERSION dependent. This bit is only defined for HPMCs; it is HVERSION dependent for TOCs and soft boots.
grv	GRs Valid. When set, the general registers are valid.
crv	CRs Valid. When set, CRs 0-16 and the EIR (CR23) are valid (also CR22 for HPMC and Soft Boot). This does not include the IPRs, the IIA queues, or the temporary registers since they have individual indicators. In addition, this bit does not indicate the validity of the IPSW for TOCs; the IPSW must always be valid for TOCs.
srv	SRs Valid. When set, the space registers are valid.
trv	Temporary Registers Valid. When set, the temporary registers (CR24 through CR31) are valid.

For each of the bits *grv*, *crv*, *srv*, *trv* in the CPU State word, a value of 1 implies that the corresponding CPU state in PIM reflects the state that would have occurred if the event had been processed as a Group 2 interruption.

The following table describes how the *iqv*, *iqf*, and *ipv* bits qualify the CPU state:

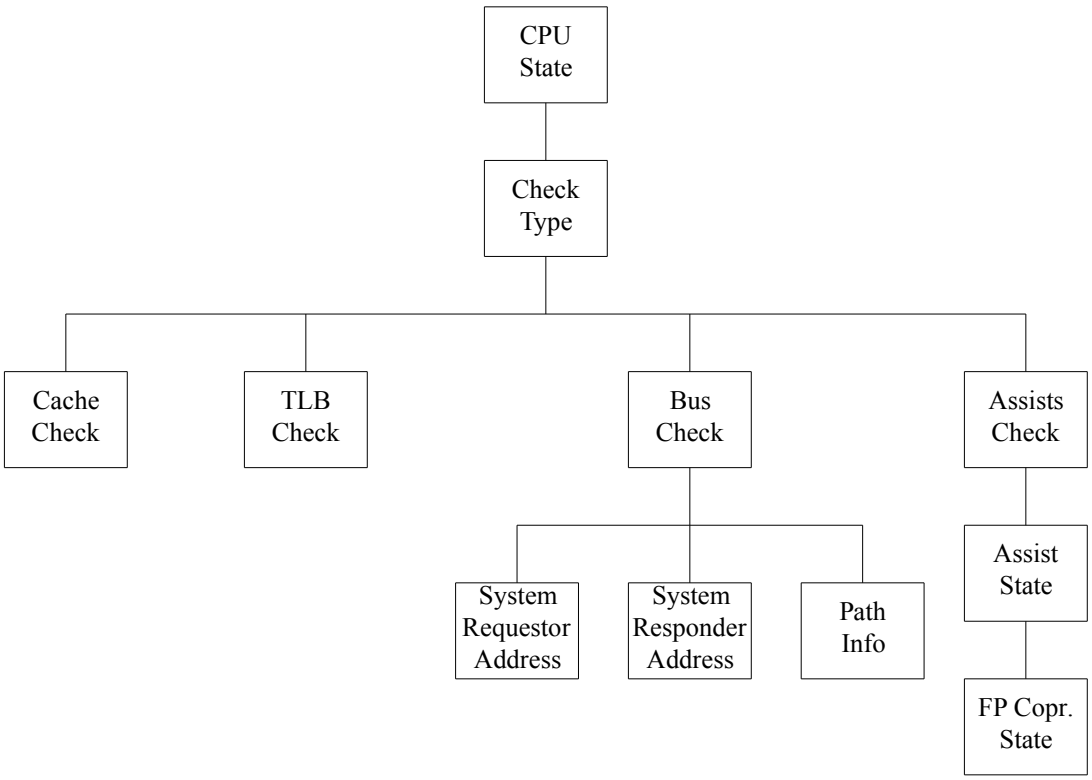
iqv	*iqf*	*ipv*	Meaning
0	0	0	The IIA queues reflect the state that would have occurred if the HPMC had been processed as a Group 1 interruption.
1	0	0	The IIA queues reflect the state that would have occurred if the HPMC had been processed as a Group 2 interruption.
1	1	0	The IIA queues reflect the state that would have occurred if the HPMC had been processed as a Group 3 interruption.
1	1	1	The IIA queues reflect the state that would have occurred if the HPMC had been processed as a Group 3 interruption and the IIR contains the instruction causing the failure.

No other combinations of these three bits are allowed. Refer to *Precision Architecture and Instruction Reference Manual* for a description of how Group 1, 2, and 3 interruptions are processed.

The IA queues are defined when the IPSW Q-bit is zero, but do not point to the failure.

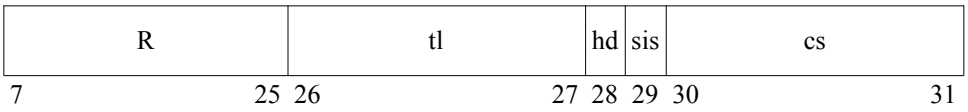
Error Parameters

A portion of PIM is used to return machine independent indicators of the failure. The nature of a failure is passed to the PDC_PIM caller by setting nonzero values in the appropriate fields. By providing as much information as possible to software, it is more likely that rebooting the system will be not be necessary. This information is provided only for HPMCs, LPMCs, and TOCs, and is presented hierarchically.



CPU State word format for HPMC

The applicable portion of the CPU State word for the HPMC option (ARG1=0) is as follows:



Field	Description
tl	Trap Lost. This field indicates which, if any, of the Group 4 interruptions were lost as a result of taking the HPMC. This field is only valid when CHECK_ISOLATED is logged.
hd	Hardware Damage. This bit is used to indicate that some processor hardware is damaged and so this processor must no longer be used, although certain minimum functionality is available. This bit is only valid when CHECK_ISOLATED or CHECK_CRITICAL is logged.
sis	Storage Integrity Synchronized. This bit is used to describe the point of storage integrity. This bit is only valid when CHECK_ISOLATED is logged.
cs	Check Severity. This field is used to determine the severity of the HPMC.

When there is hardware damage, software must avoid the following until the processor is replaced:

- Access of any of the TLB functionality.
- Access of any assist processor functionality.
- PDC calls except for PDC_PIM, PDC_CHASSIS, PDC_CONFIG, and PDC_PROC.

Encoding for the *tl* field:

tl	Meaning
0	no Group 4 interruption lost
1	the HPMC caused the loss of a higher-privilege transfer trap
2	the HPMC caused the loss of a lower-privilege transfer trap
3	the HPMC caused the loss of a taken branch trap

The check severity field, *cs*, allows OS_HPMC to determine the error severity, and thus what actions need to be taken. The interpretation of the encodings of the check severity field is as follows:

Value	Name	Description
0	CHECK_CRITICAL	An error has occurred which hardware determines to be serious enough to require a reboot. This may be because hardware has lost information about the error, or because there is no way to encode the error in PIM. This indicates a possible lack of storage integrity.
1	CHECK_TRANSPARENT	An error has occurred but has been fully corrected or circumvented in a way transparent to software. There must be synchronized storage integrity for this encoding to be logged.
2-3	CHECK_ISOLATED	An error has occurred but the PIM contents must be used by OS_HPMC to determine the error severity and decide what recovery actions are required. There must be storage integrity for this encoding to be logged, although it may not be synchronized.

If CHECK_CRITICAL is logged, all PIM error parameters other than the CPU State word are HVERSION dependent.

SUPPORT NOTE

Implementations are encouraged, when logging CHECK_CRITICAL, to set all other indicators describing the error in PIM to valid information about the error.

If CHECK_ISOLATED is logged, hardware must consider whether the error would remain isolated through recovery. For example, discovering an error in a dirty data cache line must not be logged as CHECK_ISOLATED if it might be written back to memory without signalling the error before recovery is effected.

Logging CHECK_TRANSPARENT indicates that the error was completely corrected by PDCE_CHECK; that is, OS_HPMC need only execute an RFI to resume normal system operation (provided that the CPU state in PIM is valid). Problems which require OS_HPMC recovery action, or which may have altered architectural state observable to software must not be logged as CHECK_TRANSPARENT. (Invalidating TLB entries, for example, is not always transparent, since software relies on certain translations remaining in the TLB.) All other fields in PIM must contain valid information about the error.

Logging CHECK_TRANSPARENT indicates that there is storage integrity, and that if *iqv*=1 and IPSW Q-bit=1, the IIA queues indicate the point of storage integrity

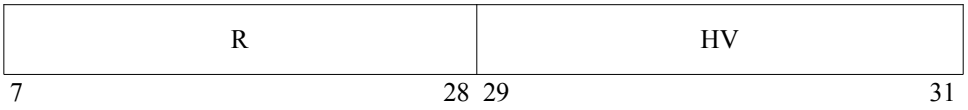
Although logging CHECK_ISOLATED indicates that there is storage integrity, nothing is asserted about the point of storage integrity. The Storage Integrity Synchronized bit, *sis*, describes the point of storage integrity. If CHECK_ISOLATED is logged and *iqv*=1, a value of one means that the point of storage integrity is at the place pointed to by the queues. In other words, all stores up to the place pointed to by the queues have been completed, and no stores at or beyond the place pointed to by the queues have been completed. If CHECK_ISOLATED is logged and *iqv*=1, a value of zero means that the point of storage integrity may be before or after the place pointed to by the queues. In this situation, the processor has stopped updating cache and memory at some point, but has then gone ahead or rolled back such that the queues do not reflect the actual point where memory updating stopped.

PROGRAMMING NOTE

Software conventions could be built to consider actions other than rebooting when the *sis* bit is zero. This might involve maintaining a log in memory with an entry for each context switch. From this log, then, it could be determined which process was running when the processor stopped storing.

CPU State word format for Soft boot

The applicable portion of the CPU State word for the Soft Boot option (ARG1=3) is as follows:



CPU State word format for TOC

The applicable portion of the CPU State word for the TOC option (ARG1=4) is as follows:

The *td* field defines the error severity due to TOC, as follows:

Value	Description
0	TOC has caused system damage and a reboot is necessary.
1	TOC did not cause any damage to the system state and no queued-up transactions in the processor module were aborted.

Detailed Error Information

For HPMCs and LPMCs, additional indicators are defined to provide more detailed information about the machine check. These indicators are valid only when certain validity bits are set.

Indicator	Condition when Valid
Check Type	Always
Cache Check	Check Type[c]=1
TLB Check	Check Type[t]=1
Bus Check	Check Type[b]=1
Assists Check	Check Type[a]=1
Assist State	Check Type[a]=1 and (Assists Check[coc]=1 or Assists Check[sc]=1)
Sys. Resp. Address	Check Type[b]=1 and Bus Check[rsv]=1
Sys. Req. Address	Check Type[b]=1 and Bus Check[rqv]=1
Path Info	Check Type[b]=1 and Bus Check[piv]=1
FP Copr. State	Check Type[a]=1, Assists Check[coc]=1, and Assist State[fps]=2

For HPMCs, CHECK_ISOLATED or CHECK_TRANSPARENT must also be logged for these indicators to be valid.

Check Type

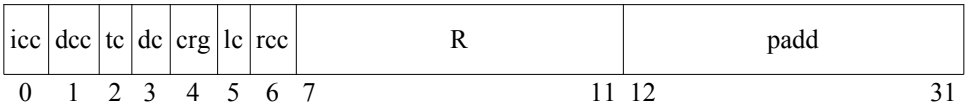
The Check Type word allows the Operating System to determine where the machine state is potentially corrupt and if functionality of the system is reduced. Each bit of the word corresponds to a major functional area of the processor which could have caused the machine check.

The format of the Check Type word is as follows:

Field	Description
c	Cache Check. If *c*=1, the Cache Check word is valid.
t	TLB Check. If *t*=1, the TLB Check word is valid.
b	Bus Check. If *b*=1, the Bus Check word is valid.
a	Assists Check. If *a*=1, the Assists Check word is valid.

Cache Check

The format of the Cache Check word is as follows:



Field	Description
icc	I-cache check. The failure is located in the I-cache.
dcc	D-cache check. The failure is located in the D-cache or a combined cache.
tc	Tag check. The failure is in the tag portion of some D-cache or combined cache line.
dc	Data check. The failure is in the data portion of some D-cache or combined cache line.
crg	Reconfiguration. Reconfiguration has occurred in the I-cache or D-cache.
lc	Line corrupt. The failure resulted in loss of tag, data, or status in some cache line.
rcc	Remote cache coherence. When set, the generation of coherent operations has been disabled; PDC_CACHE must be called before coherent operations may be issued.
padd	Physical address. The *padd* field specifies the page-aligned physical address of the failing line and is valid only if *tc*=0 and *lc*=1.

The *icc* and *crg* bits are used for reporting errors in the I-cache. The *dcc*, *tc*, *dc*, *crg*, *lc*, and *padd* bits are used for reporting errors in a D-cache or combined cache. If *dcc* is 0, *tc*, *dc*, *lc*, and *padd* must all be zero. If the *lc* bit is 1, data which may have been dirty was lost from the cache, and software action is necessary. If the *tc* bit is also 1, it is not known where in memory the line containing the lost data came from. If *tc* is 0, the *padd* field contains the physical page number of the page containing the lost data. This might result in a reboot, depending on whether this page belonged to a critical process or not. If the *lc* bit is 0, no data has been lost and no software action is required.

The encodings of various cache errors using this word is as follows:

icc	*dcc*	*tc*	*dc*	*lc*	Description
1	0	0	0	0	The error was in the I-cache.
0	1	X	X	0	The error was in some clean D-cache line. The *tc* and *dc* fields indicate whether the error was in the tag or data portion of the cache.
0	1	0	1	1	The error was in the data portion of some dirty D-cache line. This error situation signals the corruption of the data at physical address *padd*.
0	1	1	0	1	The error was in the tag portion of some D-cache line.

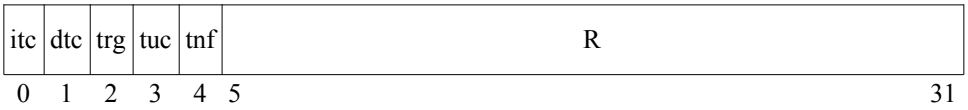
Hardware is not required to report any reconfiguration information.

SUPPORT NOTE

To increase system diagnosability, implementations are encouraged to inform the OS about corrected cache errors and cache reconfiguration.

TLB Check

The format of the TLB Check word is as follows:



Field	Description
itc	ITLB Check. The failure is located in the ITLB.
dtc	DTLB Check. The failure is located in the DTLB or a combined TLB.
trg	Reconfiguration. Reconfiguration has occurred in the ITLB or DTLB.
tuc	TLB Unchanged. When set, PDCE_CHECK has not invalidated, removed, or initialized all or part of the TLB system.
tnf	TLB Nonfunctional. When set, the TLB subsystem (including the space registers) is nonfunctional and access to it may cause unpredictable results.

When the TLB subsystem is nonfunctional, software must avoid the following:

- Execution of any of these instructions: BLE, IDTLBA, IDTLBP, IITLBA, IITLBP, LDSID, LHA, LPA, MFSP, MTSP, PDTLB, PDTLBE, PITLB, PITLBE, PROBER, PROBERI, PROBEW, PROBEWI, PDC, FDC, FIC.
- The setting to 1 of the PSW C, P, and D-bits.
- Execution of MTCTL or MFCTL which references any of these control registers:
  - CR 8 (PID 1)
  - CR 9 (PID 2)
  - CR 12 (PID 3)
  - CR 13 (PID 4)
  - CR 20 (Interruption Space Register)
  - CR 21 (Interruption Offset Register)

Since the FDC instruction must not be used when the TLB is nonfunctional, OS_HPMC must flush the entire cache using FDCE instructions if it wants to flush any item from the data cache.

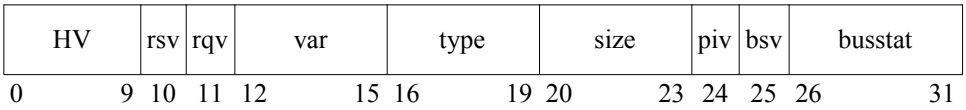
Hardware is not required to report any reconfiguration information.

SUPPORT NOTE

To increase system diagnosability, implementations are encouraged to inform the OS about any reconfiguration in the TLB(s).

Bus Check

The format of the Bus Check word is as follows:



Field	Description
rsv	when set, the System Responder Address word is valid
rqv	when set, the System Requestor Address word is valid
var	indicates the variant of the transaction in which the error occurred
type	indicates the type of the transaction in which the error occurred
size	indicates the size of the transaction in which the error occurred
piv	when set, the Path Info word is valid
bsv	error severity for bus errors
busstat	encodings for architected bus errors

Encoding for the *var* field:

Variant	Description
0	Unknown or illegal variant
1	DFLT
2	INV
3	SH
4	PVT
5	SH_PVT
6	COH
7	Reserved
8-15	Bus Spec. Dep.

Encoding for the *type* field:

Type	Description
0	Unknown or illegal type
1	READ
2	WRITE
3	CLEAR
4	READ_REQ
5	READ_RESP
6	CLEAR_REQ
7	NULL
8	PDC
9	FDC
10	FIC
11	SYNC
12	PDTLB
13	PITLB
14-15	Reserved

Encoding for the *size* field:

Size	Description
0	Unknown or illegal size
1	1 byte
2	2 byte
3	4 byte
4	Reserved
5	16 byte
6	32 byte
7	64 byte
8	128 byte
9	256 byte
10	512 byte
11	1024 byte
12	2048 byte
13	4096 byte
14	Reserved
15	No size[1]

Notes:

1. Required for PDC, FDC, FIC, SYNC, PDTLB, PITLB, and NULL transactions. Not allowed for other transactions.

Encoding for the *bsv* field:

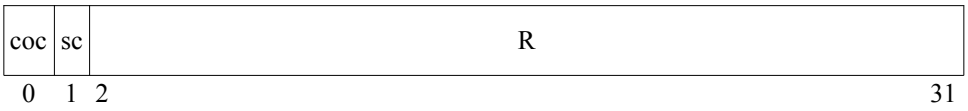
bsv	Severity
0	Fatal
1	Soft

Encoding for the *busstat* field:

busstat	Error
0	ERR_DEPEND
1	ERR_UNIMPL
2	Reserved
3	ERR_MODE_RS
4	ERR_ERROR_RQ
5	ERR_PARITY_RS
6	ERR_PROTOCOL_RQ
7	ERR_ADDRESS_RQ
8-12	Reserved
13	HV
14	Reserved
15	ERR_BUS_RQ
16-23	HV
24-49	Reserved
50	ERR_RESPONSE
51	ERR_BUS_RS
52	ERR_ERROR_RS
53	ERR_PARITY_RQ
54	ERR_PROTOCOL_RS
55	ERR_ADDRESS_RS
56	ERR_MODE_RQ
57-58	Reserved
59	ERR_TIMEOUT
60	ERR_RETRY
61	Reserved
62	ERR_IMPROP
63	Reserved

Assists Check

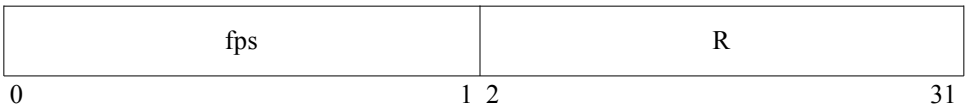
The format of the Assists Check word is as follows:



Field	Description
coc	Coprocessor Check. If *coc*=1, bits 0..15 of the Assist State word are valid.
sc	SFU Check. If *sc*=1, bits 16..31 of the Assist State word are valid.

Assist State

The format of the Assist State word is as follows:



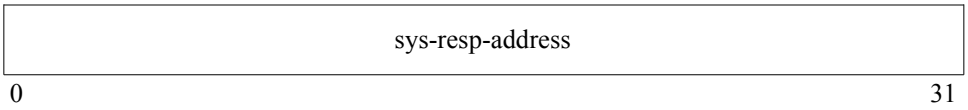
The *fps* field defines the floating-point coprocessor (uid=0 or uid=1) state:

Value	Description
0	The coprocessor is functional but its state is invalid and is not saved in PIM.
1	The coprocessor has failed and its state is not saved in PIM.
2	The coprocessor has failed but its state has been saved in PIM.
3	Reserved

When the floating-point coprocessor has failed (*fps* = 1 or 2), software must not execute any floating-point instruction until the CCR bit 0 is cleared.

System Responder Address

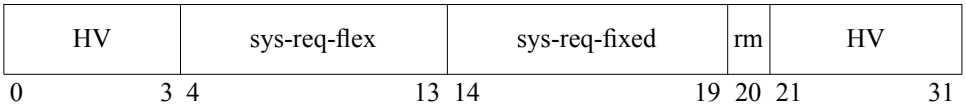
The System Responder Address word identifies the system responder of a failed operation; its format is as follows:



The number of bits in this field which are valid depends on the operation size. Valid bits are 0..*n*, where $n = 31 - \log_2 (size)$; the remaining bits are HVERSION dependent. Since the largest system operation size is 4 Kbytes, bits 0..19 are always valid when *rsv*=1.

System Requestor Address

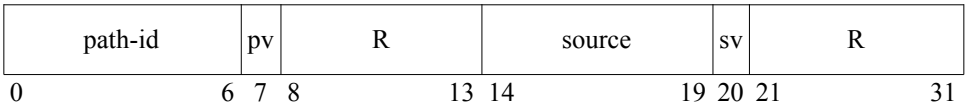
The System Requestor Address word identifies the system requestor of a failed operation; its format is as follows:



Field	Description
sys-req-flex	System requestor's flex address
sys-req-fixed	System requestor's fixed address
rm	Validation of the *sys-req-flex* field

Path Info

The format of the Path Info word is as follows:

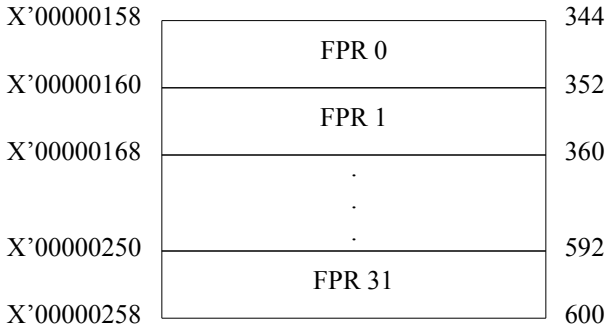


The *path-id* field indicates the transaction path used by the operation for which the error is logged and it is qualified by the *pv* bit. The *path-id* field is valid when *pv*=1.

The *source* field is the fixed address of the source module of the operation. In a cache-coherent operation, a cache-coherent module may respond, rather than the addressed system responder module. It is the fixed address of this third party module which becomes the source address of the operation. Logging of the source address is optional; the *sv* bit qualifies the *source* field.

Floating-Point Coprocessor State

The floating-point coprocessor state is returned in the following format:



If software wants the address of these registers in memory to occur at doubleword-aligned boundaries, it must adjust the word-aligned *memaddr* parameter in the PDC_PIM procedure to be doubleword aligned.

PDC_POW_FAIL (index 1)

Purpose: To perform whatever HVERSION dependent steps are necessary to prepare the system for powerfail.

Arguments:

Description	ARG1
Prepare for powerfail	0

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
0	OK The call completed normally and the procedure detected no error. No powerfail warning is in effect. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.

Description: This procedure is called by the operating system when it receives a power failure interrupt and after it completes its own powerfail preparation. PDC_POW_FAIL does whatever HVERSION-dependent preparation is necessary and then waits idly until primary power fails. In the idle state, the procedure must not generate any bus transactions.

If no powerfail warning is in effect at the time of the call, the PDC_POW_FAIL procedure returns to the caller.

The "**Prepare for powerfail**" option (ARG1=0) checks the BUS_POW_WARN signal on the central bus. If BUS_POW_WARN is asserted, PDC_POW_FAIL prepares the system for the loss of primary power. After preparation is completed, it enters an idle loop. If BUS_POW_WARN is not asserted, PDC_POW_FAIL simply returns to its caller.

The stack space available for use by PDC_POW_FAIL is 512 bytes of memory, in contrast to the 7 Kbytes available for other PDC and IODC procedures.

The existence of the PDC_POW_FAIL procedure is HVERSION dependent. Processors in systems which support powerfail recovery must provide this procedure; processors in systems which do not support powerfail recovery must not provide this procedure.

PDC_PROC (index 16)

Purpose: To stop or rendezvous the currently executing processor in a multiprocessor system.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4	ARG5	ARG6	ARG7
Stop processor	0	R	R	R	R	R	R
Rendezvous processor	1	R	R	R	R	R	R

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: The "**Stop processor**" option (ARG1=0) causes the processor currently executing to stop. Bus requestorship is disabled, the PSW I-bit is 0, and the module no longer participates in cache coherence protocols. It is the callers responsibility to ensure that any required data is flushed from the cache prior to calling the Stop Processor option.

The "**Rendezvous processor**" option (ARG1=1) causes the processor currently executing to enter the PDCE_RESET rendezvous code. The processor must meet all the boundary conditions specified in the OS_RENDEZ interface and in the PDCE_RESET definition for a rendezvousing processor. This option must be implemented by category B (multiprocessor) processor modules, and is HVERSION dependent for category A (uniprocessor) modules.

PDC_PSW (index 21)

Purpose: To manage the default value of configurable PSW bits for a processor.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4
Return Mask	0	R_addr	R	R
Return Defaults	1	R_addr	R	R
Set Defaults	2	state	R	R

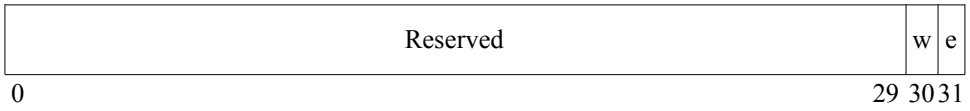
Returns:

Description	RET[0]
Return Mask	mask
Return Defaults	defaults
Set Defaults	--

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: The **"Return Mask"** option (ARG1=0) returns, in *mask*, a mask indicating which default PSW bits are implemented by the processor. The format of *mask* is:



A corresponding default PSW bit is implemented if and only if the mask bit is 1.

The **"Return Defaults"** option (ARG1=1) returns the current default PSW values of the processor in *defaults*. The format of *defaults* is the same as that described above for *mask*. The *w* field indicates the default width of the processor. The *w* field also determines whether the External Interrupt Request (EIR) register is treated as a right-justified 32-bit register or a full 64-bit register. The *e* field indicates the default endianness of the processor. Both bits determine how the PSW W-bit and E-bit will be set on an interruption. Only those bits whose *mask* value (from the **"Return Mask"** option) is 1 are valid.

The **"Set Defaults"** option (ARG1=2) sets the default PSW values of the processor to the value specified in *state*. The format of *state* is the same as that described above for *mask*. Only those bits whose *mask* value (from the **"Return Mask"** option) is 1 will be affected.

The default PSW values are per-processor resources and must be maintained as non-volatile state.

PDC_SOFT_POWER (index 23)

Purpose: To allow software control of the power off feature.

Arguments:

Description	ARG1	ARG2	ARG3
Return Information	0	R_addr	R
Enable/Disable Soft Power	1	R_addr	Power_control

Returns:

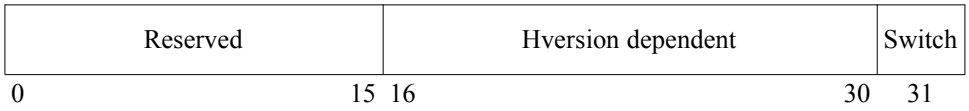
Description	RET[0]	RET[1]	RET[2]
Return Information	Poll_addr	R	R
Enable/Disable Soft Power	R	R	R

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: The **"Return Information"** option (ARG1=0) returns the location of the Soft Power Register, which may be read to determine the current power switch setting. The address is returned in *poll_addr*, which is a 32 bit physical address. This location may be read, in turn, to determine the power switch setting. A zero in bit position 31 indicates that the switch is in the off position. A one in bit position 31 indicates that the switch is in the on position.

The full layout of the Soft Power Register is shown in the following diagram.



The **"Enable/Disable Soft Power"** option (ARG1=1) allows the OS to enable or disable software control of the power off feature. At power on and reset the feature is disabled. That is, power is under hardware control. When **"Enable/Disable Soft Power"** is called with Power_control (ARG3) set to one, the power switch is placed under software control.

ENGINEERING NOTE

For implementations which use a LED to indicate the Soft Power feature, Calling **"Enable/Disable Soft Power"** with Power_control set to one causes the LED to go from flashing mode to solid on mode.

When **"Enable/Disable Soft Power"** is called with Power_control set to zero, the power switch is placed under direct hardware control. If the hardware switch is in the off position, the system is

immediately powered off. If the switch is in the on position, software control is disabled, and and the PDC call returns to the OS.

ENGINEERING NOTE

For implementations which use a LED to indicated the Soft Power feature, Calling **"Enable/Disable Soft Power"** with Power_control set to zero when the power switch is in the on position causes the LED to go from solid on mode to flashing mode.

PDC_STABLE (index 10)

Purpose: To provide access to Stable Storage.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4
Read data	0	staddr	memaddr	count
Write data	1	staddr	memaddr	count
Return size	2	R_addr	HV	---
Verify contents	3	HV	HV	---
Initialize	4	HV	HV	---

Returns:

Description	RET[0]
Read data	---
Write data	---
Return size	size
Verify contents	---
Initialize	---

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
1	Correctable error The call completed normally and the returned results are valid. The procedure encountered an error which it was able to correct completely. Returned only by options ARG1=0, 1, 3, and 4. CONDITIONAL. Must be used if PDC_STABLE performs error recovery.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-5	Invalid Stable Storage contents Returned only by options ARG1=0, 1, 3, and 4. REQUIRED.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. CONDITIONAL. Must be returned by ARG1=0 or 1 if *staddr*+*count* > *size*. Otherwise, the procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: **Stable Storage** is used to maintain system parameters during power outages. It is required to retain its contents even if the card containing it is removed from the backplane. Stable Storage is used during boot. It contains the paths to the console and boot devices.

Stable Storage is a system-wide resource shared by all processors in a multiprocessor system. Software must ensure that at most one call to PDC_STABLE is in progress at any one time.

The integrity of the storage must be guaranteed. Reliability must be such that an undetected error will occur only once in the lifetime of a million machines.

Stable Storage must have a minimum lifetime of 10,000 write cycles.

The "**Read data**" option (ARG1=0) transfers *count* bytes from Stable Storage address *staddr* to memory address *memaddr*. *count* is an unsigned 32-bit integer which is a multiple of four. *staddr* and *memaddr* must be word aligned. The call must return -10 if *staddr+count > size*.

Reads must validate data integrity for the bytes being read. If a checksum algorithm is used, this may involve validating all of Stable Storage. If the data integrity check fails, the call must return -5 and (if possible) the potentially bad data.

The "**Write data**" option (ARG1=1) transfers *count* bytes from memory address *memaddr* to Stable Storage address *staddr*. *count* is an unsigned 32-bit integer which is a multiple of four. *staddr* and *memaddr* must be word aligned. The call must return -10 if *staddr+count > size*.

Writes must not mask errors in Stable Storage. Before a write is attempted, PDC_STABLE must check the data integrity of the words being modified. If the check fails, PDC_STABLE must not attempt to write any data: instead, it must return -5. If the check succeeds, the write can proceed. After the data is written, PDC_STABLE must also verify that the write took place correctly. If the write did not succeed, -3 must be returned.

If a write to Stable Storage is interrupted due to a powerfail, reset, or TOC, and if PDC cannot guarantee that the write completed, then it must ensure that data integrity checks fail on subsequent accesses to Stable Storage.

ENGINEERING NOTE

A recommended method to check the validity of Stable Storage is to include a checksum in a non-architected tertiary state storage area. This checksum should be designed such that a Stable Storage filled with all zeroes or all ones will not generate a valid checksum.

If a write to Stable Storage is interrupted by a powerfail, reset, or TOC, hardware implementations should attempt to limit the extent of damage to the words that were being modified. This rule applies except for genuine hardware failures or sudden power failures. For such failures, the extent of damage cannot be predicted.

The "**Return size**" option (ARG1=2) returns the number of bytes in the processor's Stable Storage. *size* is the number of contiguous bytes implemented in Stable Storage starting from *staddr*=0. *size* is an unsigned 32-bit integer which is a multiple of four.

The "**Verify contents**" option (ARG1=3) verifies that the Stable Storage contents are valid.

The "**Initialize**" option (ARG1=4) sets the architected contents of Stable Storage to zero and initializes the validity indicator. Unarchitected areas in Stable Storage must be zeroed individually when required.

PROGRAMMING NOTE

An algorithm to restore Stable Storage in the event of a failure follows:

1. Copy out the entire contents of Stable Storage into memory.
2. Fix all the bad values by correcting the copied values in memory.
3. Call the "Initialize" option to zero Stable Storage.
4. Write the good copy from memory to Stable Storage.
5. Call the "Verify contents" option to check for successful completion.

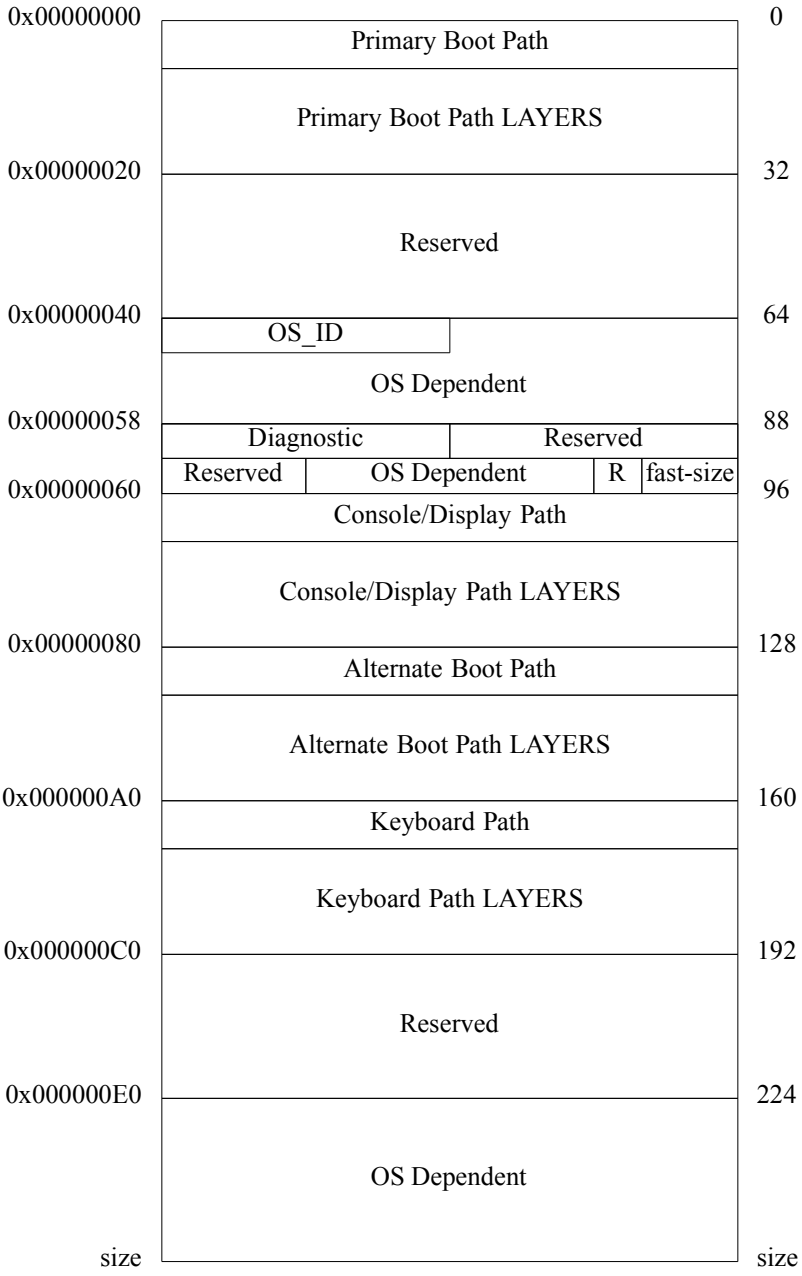
To protect against a powerfail after the "Initialize" option has zeroed Stable Storage, but before the new values are written back, software should back up the contents on disk before zeroing, or be able to reconstruct Stable Storage from the all zero state.

ENGINEERING NOTE

Specific products may choose to implement multiple copies of Stable Storage to increase the fault tolerance of the system, but this must be transparent to the callers of PDC_STABLE.

Data Format of Stable Storage

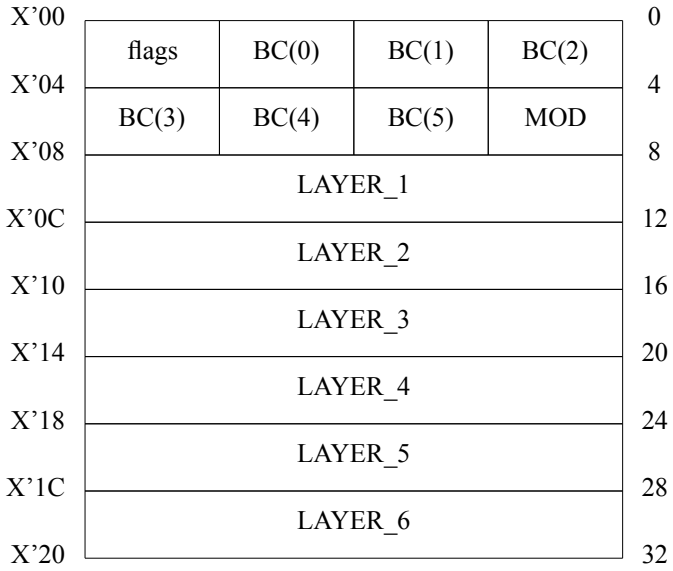
The format of Stable Storage is as follows:



A minimum of 96 bytes of Stable Storage is required. Providing more than 96 bytes of Stable Storage is optional, but it must be used in the architected way if provided. Failure to provide the optional locations from 96 to 192 results in the loss of certain functionality during boot. (For example, ISL would not be able to set the console path.)

Primary Boot Path

The format of the Primary Boot Path is as follows:



The format of *flags* is as follows:

The autoboot and autosearch bits, *ab* and *as* respectively, select the mechanism used to locate the boot device. The *timer* field is used by PDC to initialize the value of its boot timer. If *timer* is 0, PDC initializes the boot timer to an HVERSION-dependent default value. Otherwise, PDC initializes its boot timer to 2^{timer} seconds.

The values of BC(0) through BC(5) specify the bus converter routing to the specified boot module. Values of 0 through 63 specify the fixed field of the bus converter port's HPA. The values 64-127 are reserved. The values 128-255 are null values, and are ignored in the path specification.

If only N bus converters are specified in the path to the specified boot module, the leading bytes of the path specifier, BC(0) through BC(5-N), are null. In that case, BC(6-N) specifies the routing through the bus converter closest to the processor.

The value of MOD is the fixed field of the specified module. The values 64-127 are reserved. The values 128-255 are null values, and indicate that the path has not been specified.

The six-word LAYERS block is used to describe the portion of the path to a device that is beyond the module and/or to contain device-dependent information. The path specification is based on a model that assumes a set of layers beyond the module. A layer is defined as a series of entities which are separately addressable and exist at the same level in a hierarchical tree structure.

Two types of entity make up the tree structure in the layers beyond the module. **Devices** form the leaves in the tree structure. **Controllers** form the intermediate entities between the module and devices.

ENGINEERING NOTE

For example, an HP-CIO Adapter always has layers beyond the module. The first layer contains the device adapters on the CIO bus. If that device adapter is an HPIB adapter, the next layer might include a disk controller on the HPIB bus.

Each entity within a layer is identified by a 32-bit number. That number should have some physical

correspondence to the address of the entity within the layer. The number should be easy for the operator to associate with the entity, for example, via switch positions.

If there are N layers beyond the module, the words LAYER_1 through LAYER_N specify the addresses of entities in those layers, in order, starting with the layer closest to the module. There can be a maximum of six layers beyond the module. If there are less than six layers beyond the module, the words after the ones used for layers are device dependent. There is no delimiter to mark the end of the layers and the start of the device-dependent words. It is expected that the boundary will be implicitly understood by the routines that are using the path. If there are no layers beyond the module, all six words are device dependent.

OS Dependent

There are three OS Dependent fields in Stable Storage. The first field is 24 bytes and is located at X'40. The second field is two bytes and is located at X'5D. The third field is at X'E0 and occupies the rest of Stable Storage up to the *size* value returned by the "Return Size" option.

The first halfword of the first OS Dependent area (at X'40) is used to store an identifier called OS_ID. OS-dependent use of Stable Storage and Non-Volatile Memory is qualified by OS_ID.

Values of OS_ID are assigned by Hewlett-Packard's Computer Systems Architecture team.

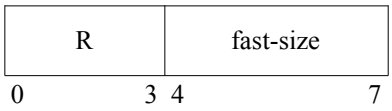
Data read from either of the OS-dependent areas of Stable Storage or the OS Panic Information area of Non-Volatile Memory must be interpreted in the context of the value of OS_ID.

The operating system should ensure that the value of OS_ID is appropriate and that the OS-dependent areas are initialized to proper default values.

The format of the second OS dependent area (at X'5D) and the third OS dependent area (at X'E0) depends on OS_ID.

Fast-size

Byte X'5F is defined as follows:



where *fast-size* specifies the amount of memory that PDCE_RESET initializes and tests, as follows:

fast-size	Memory Tested
0	256 KB
1	512 KB
2	1 MB
3	2 MB
4	4 MB
5	8 MB
6	16 MB
7	32 MB
8	64 MB
9	128 MB
A	256 MB
B	512 MB
C	1 GB
D	2 GB
E	all
F	all

Values of *fast-size* other than 'E' or 'F' specify the amount of contiguous memory to be tested; in these cases, if the amount of contiguous memory on the local bus is less than the amount specified by *fast-size*, all contiguous memory is tested and no error is indicated. The memory need not be interleaved.

When a memory module's SPA is larger than *fast-size*, PDCE_RESET must configure the memory module, but must only initialize and test up to *fast-size*. That is, any transactions beyond *fast-size*, but within the memory's SPA must be slave acknowledged.

The values 'E' and 'F' allow all local memory, contiguous or noncontiguous, to be tested by PDC.

Console/Display Path, Alternate Boot Path, and Keyboard Path

Except for the *flags* field, the format of the Console/Display, Alternate Boot, and Keyboard Path areas are the same as defined for the Primary Boot Path. The *flags* field for the Console/Display Path contains a *timer* field the same as the Primary Boot Path, but not the *ab* or *as* bits. The *flags* field in the Alternate Boot Path and Keyboard Path is reserved.

ENGINEERING NOTE

Stable Storage, which can be modified by software, must have an initial value when shipped from the factory. Each processor design group is responsible for defining initial Stable Storage values for that processor. In the absence of more specific information, all bytes in Stable Storage should be initialized to 0, except bytes X'07, X'67, X'87, X'A7 which should be set to X'FF (path not specified) and byte X'5F which should be set to X'0F (*fast-size* = all).

PDC_SYSTEM_MAP (index 22)

Purpose: To locate and identify modules in a system which reside at fixed I/O addresses, and which do not self identify.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4
Find Module	0	R_addr	mod_path	mod_index
Find Address	1	R_addr	mod_index	addr_index
Translate Path	2	R_addr	mod_path	R

Description	ARG5	ARG6	ARG7
Find Module	R	R	R
Find Address	R	R	R
Translate Path	R	R	R

Description	RET[0]	RET[1]	RET[2]	RET[3]
Find Module	mod_addr	mod_pgs	add_addrs	
Find Address	mod_addr	mod_pgs		
Translate Path	mod_addr	mod_pgs	add_addrs	mod_index

Value	Description
0	OK. The call completed normally and the procedure detected no error. REQUIRED
-2	Non-existent option. ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error. An error of unspecified type prevented the call from completing normally. CONDITIONAL. Must be used if indeterminate errors can be detected.
-5	Module not found. No module exists which corresponds to the LAYER structure or module index specified. REQUIRED.
-6	Address range not found. No address range exists which corresponds to the address index specified. REQUIRED. Returned only by option ARG1=1.
-10	Invalid Argument An argument other than ARG0 or ARG1 was invalid OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN detected. CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: The PDC_SYSTEM_MAP procedure is called by the operating system when it is configuring the IO system immediately after a boot. It is used to locate and identify modules which can not provide IODC.

"Find Module" returns information about the fixed address module indexed by *mod_index*. Fixed address modules are indexed from 0 to N-1, where N is the number of fixed address modules in a system. Upon entry to "Find Module" *mod_path* (ARG3) contains a pointer to a LAYER structure (see PDC_STABLE), but the LAYER structure need not be initialized to any particular value. If the option finds a fixed address module, it returns the first page of the first address range in *mod_addr* (RET[0]) and the number of IO pages within the address range in *mod_pgs* (RET[1]). In addition, it places the physical path of the module in the LAYER structure pointed to by *mod_path* (ARG3). If the module contains multiple discontiguous address ranges, a non-zero number is returned in *add_addrs* (RET[2]) which is the number of additional address ranges. The value returned in *mod_addr* is the address passed to PDC_IODC to get module type and version

information. If there are no fixed address modules at index *mod_index,* "Find Module" will return with a status value of -5.

"Find Address" returns information about subsequent address ranges for modules with multiple address ranges. Upon entry to "Find Address" *mod_index* contains the value used in the *mod_index* argument of the previous "Find Module" or "Find Address." *addr_index* contains the index of the address range for which information is requested. If the option finds an address range corresponding to that module and address index, the address of the first page is returned in *mod_addr* (RET[0]), and the number of pages in the address range is returned in *mod_pgs* (RET[1]). If there is no module corresponding to *mod_index,* "Find Address" will return with a status value of -5. If there is no address range corresponding to *addr_index*, "Find Address" will return with a status value of -6.

The "Translate Path" option returns module information for the module whose path is described by the LAYER structure pointed to by *mod_path* (ARG3). The values returned in RET[0] through RET[2] are exactly as described in "Find Module". In addition "Translate Path" returns the module index of the module in *mod_index*. For modules with multiple discontiguous address ranges, this can be used with "Find Address." If the path in the LAYER structure does not correspond to an actual module, "Translate Path" returns with a status of -5.

PROGRAMMING NOTE

The following code describes a high level OS algorithm which could use this PDC procedure on any existing (9000/700, 9000/800, 3000/900) platform or currently planned future platforms.

```
Find monarch processor and memory controller;
If (processor.HVERSION is Snakes) {       /* cannot use PDC call existence */
   Find fixed devices using PDC_MEMORY_MAP; /* because of Emerald conflicts */
};
Else {   /* we're either on a new system or old s800 or s900 */
   module_index = 0;
   if (status(PDC_SYSTEM_MAP("Find Module",module_index))!=-1) {
      /* we are on a new system with PDC_SYSTEM_MAP */
      while (status!=-5) {
         Add a new module to IO table;
         if (add_addrs) {
            for (address_index = 1; address_index++; address_index <= add_addrs) {
               PDC_SYSTEM_MAP("Find Address", address_index);
               Add a new address range to module;
            };
         };
         module_index++;
         PDC_SYSTEM_MAP("Find Module",module_index)
      }   /* continue loop while status != -5 */
   };   /* when we get here either we've found all fixed or we are s8/900 */
   Use old s800 or s900 algorithm to find configurable modules;
};
```

PDC_TLB (index 19)

Purpose: To manage hardware TLB miss handling.

Arguments:

Description	ARG1	ARG2	ARG3	ARG4	ARG5	ARG6	ARG7
Return Parameters	0	R_addr	R	R	R	R	R
Set up miss handling	1	R_addr	base_addr	table_size	set_state	R	R

Returns:

Description	RET[0]	RET[1]
Return Parameters	min_size	max_size
Set up miss handling	state	R

Status:

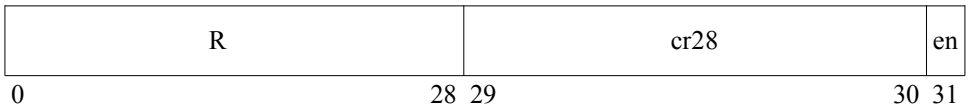
Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.

Description: The "**Return Parameters**" option (ARG1=0) returns two parameters that characterize the processor's hardware TLB miss handling.

min_size and *max_size* are the minimum and maximum sizes, in bytes, of the hardware-visible page table supported by the processor. *min_size* and *max_size* are 32-bit unsigned integers. *min_size* and *max_size* must be within the range 4096 to 2^{30}, and be a power of two.

The "**Set up miss handling**" option (ARG1=1) sets up hardware TLB miss handling. *base_addr* specifies the starting physical address of the hardware-visible page table. *table_size* specifies the size, in bytes, of the hardware-visible page table. *base_addr* must be *table-size* aligned.

The *set_state* argument controls the state of hardware TLB miss handling. The format of the *set_state* argument is the following:



The *en* bit determines whether hardware miss handling is enabled (*en* = 1) or disabled (*en* = 0). All processors must support enabling and disabling of hardware TLB miss handling. The *cr28* field determines how the processor's control register 28 is updated when the hardware miss handler fails to insert a translation and traps to software. The values of the *cr28* field are as follows:

Value	Description
0	Pointer to current page table entry
1	Reserved
2	Value from the next page table entry field of current entry
3	Value from word 3 of the 16-byte line containing the current entry

Support for *cr28* values other than 0 is optional. The *state* return value, which has the same format as *set_state*, indicates the closest *cr28* value which the hardware supports. Even if the hardware does not support the requested *cr28* value, a call made with *en* = 1 will enable hardware miss handling with the *cr28* value as returned in *state*. In particular, if PDC_TLB is called with *cr28* = 1, hardware miss handling will be enabled with a *cr28* value of 0, and 0 will be returned in the *cr28* field of *state*.

When called with the *en* bit = 0, the *cr28* field of *set_state*, and the *base_addr* and *table_size* arguments are ignored.

Making a PDC_TLB call with different values of *base_addr* and/or *table_size* than were used in a previous call without first disabling hardware miss handling is allowed.

The results of calling PDC_TLB become effective immediately upon turning on either of the PSW C or D bits.

PROGRAMMING NOTE

Because there is no mechanism to read the current values of *base_addr*, *table_size*, and *set_state*, callers need to maintain these values for themselves.

PDC_TOD (index 9)

Purpose: To read, set, and calibrate the Time-Of-Day (TOD) clock.

Arguments:

Description	ARG1	ARG2	ARG3
Read TOD	0	R_addr	HV
Set TOD	1	tod_sec	tod_usec
Calibrate timers	2	R_addr	HV

Returns:

Description	RET[0]	RET[1]	RET[2]	RET[3]
Read TOD	tod_sec	tod_usec	R	R
Set TOD	---	---	---	---
Calibrate timers	calib_0	calib_1	TOD_acc	CR_acc

Status:

Value	Description
3	Call completed with a warning. An error of unspecified type occurred, but the call completed correctly. OPTIONAL. The procedure need not report warning conditions.
0	OK The call completed normally and the procedure detected no error. REQUIRED.
-2	Nonexistent option ARG1 did not correspond to an option provided by the procedure. REQUIRED.
-3	Cannot complete call without error An error of unspecified type prevented the call from completing correctly. CONDITIONAL. Must be used if indeterminate errors can be detected.
-10	Invalid argument An argument other than ARG0 or ARG1 was invalid. OPTIONAL. The procedure need not check arguments for correctness.
-12	Assertion of BUS_POW_WARN signal detected CONDITIONAL. Must be used if the procedure cannot satisfy the powerfail budget.
-13	Time of day invalid Returned only by option ARG1=0. CONDITIONAL. Must be used if the implementation has a way to tell if the clock is invalid. It is strongly recommended that all implementations which consider it important to have a valid clock provide a mechanism to detect an invalid clock.

Description: The Time-Of-Day Clock gives an absolute measure of system time. The TOD clock must be secondary powered and must preserve the system time on a primary powerfail.

The Time-Of-Day Clock is a system-wide resource. Software must guarantee that at most one call to PDC_TOD is in progress at any one time.

The "**Read TOD**" option (ARG1=0) returns two parameters which specify the elapsed time since 00:00:00 GMT, January 1, 1970. The value of *tod_sec* is interpreted as a number of seconds, and the value of *tod_usec* as microseconds. *tod_usec* is normalized to be less than 1,000,000. Both *tod_sec* and *tod_usec* are unsigned 32-bit integers.

"Read TOD" must return 0 for any least significant portion of *tod_usec* which is not accurate.

The "**Set TOD**" option (ARG1=1) is used to set the TOD clock. The meanings and format of *tod_sec* and *tod_usec* are the same as in the "Read TOD" option. Callers must normalize *tod_usec* to a value less than 1,000,000, and adjust the *tod_sec* parameter appropriately.

If the caller passes a value of *tod_usec* greater than or equal to 1,000,000 then the PDC_TOD procedure can optionally return -10 or set the TOD to an HVERSION-dependent value and return 0.

"Set TOD" may optionally discard any least significant portion of *tod_usec*.

The "**Calibrate timers**" option (ARG1=2) is used to calibrate the Interval Timer (CR16). It returns *calib_0* and *calib_1*, a double-precision floating-point value that gives the frequency of the Interval Timer in megahertz. *calib_0* and *calib_1* form a double-precision (64-bit) floating-point value. The first 32-bits (one sign bit, 11 exponent bits, and 20 most significant bits of the fraction field) are contributed by *calib_0*, the remaining 32-bits of the fraction field are defined by *calib_1*. The accuracies of the TOD clock and the Interval Timer are specified by *TOD_acc* and *CR_acc*, respectively. Both of these parameters are unsigned 32-bit integers representing clock accuracy in parts per billion.

PROGRAMMING NOTE

It is recommended that software call the "Calibrate timers" option to determine which clock (TOD clock or CR16 Interval Timer) is more accurate. Depending on the relative accuracies of the two clocks and product requirements, the frequency and mechanism of synchronizing the two clocks can be determined.

A worst case resolution of 1 sec for TOD is required. There is no guarantee that the *tod_usec* parameter is significant for either the "Read TOD" or "Set TOD" options.

There is no limit on the duration of a PDC_TOD call.

The "Read TOD" option must adjust the time returned from the TOD clock if the time from the read to the end of the call is greater than half of the worst case resolution for TOD.

The "Set TOD" option must adjust the time stored in the TOD clock if the time from the beginning of the call to the completion of the write is greater than half of the worst case resolution for TOD.

This page intentionally left blank