Intel® Itanium® Architecture Software Developer's Manual Specification Update

October 2007

Document Number: 248699-011

Contents

Preface 5
Summary Table of Changes 6
Specification Changes 7
Specification Clarifications 34
Documentation Changes 41

Revision History

Document Number	Version Number	Description	Date
248699	-011	• Specification changes 1-12, Specification Clarifications 1-10, and Document Change 1.	October 2007
248699	-001- -010	• Changes from previous Software Developer's Manual Specification Updates were incorporated into version 2.2 of the *Intel® Itanium® Architecture Software Developer's Manual* January 2006.	January 2006

§§

1 Preface

This document is an update to the specifications contained in the Affected Documents/ Related Documents table below. This document is a compilation of device and documentation errata, specification clarifications, and changes. It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools.

Information types defined in Nomenclature are consolidated into the specification update and are no longer published in other documents.

This document may also contain information that was not previously published.

1.1 Affected Documents/Related Documents

Title	Document #
Intel® Itanium® Architecture Software Developer's Manual, Revision 2.2, Volume 1: Application Architecture	245317-005
Intel® Itanium® Architecture Software Developer's Manual, Revision 2.2, Volume 2: System Architecture	245318-005
Intel® Itanium® Architecture Software Developer's Manual, Revision 2.2, Volume 3: Instruction Set Reference	245319-005

1.2 Nomenclature

Specification Changes are modifications to the current published specifications for Intel® Itanium® architecture processors. These changes will be incorporated in the next release of the specifications.

Specification Clarifications describe a specification in greater detail or further explain a specification's interpretation. These clarifications will be incorporated in the next release of the specification.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the *Intel® Itanium® Architecture Software Developer's Manual*.

2 Summary Table of Changes

The following tables indicate the specification changes and specification clarifications that apply to the *Intel® Itanium® Architecture Software Developer's Manual*.

2.1 Specification Changes

No.	Page	SPECIFICATION CHANGES
1	7	Illegal VAC/VDC combinations and IIPA requirements
2	8	Resouce Utilization Counter
3	19	PAL_VP_INIT and VPD.vpr changes
4	18	New PAL_VPS_RESUME_HANDLER to indicate RSE Current Frame Load Enable setting at the target instruction
5	19	PAL_VP_INIT_ENV Implementation-specific Configuration Option
6	19	Increase in minimum number of virtual address bits
7	20	PAL_MC_ERROR_INFO health indicator
8	23	New implementation-specific bit fields for PAL_MC_ERROR_INJECT
9	23	MOV-to-PSR.L Reserved Field Checking
10	23	Virtual Machine Disable
11	26	Removal of pal_proc_vector argument from PAL_VP_SAVE and PAL_VP_RESTORE
12	26	Variable Frequency Mode Additions to ACPI P-states
13	29	PAL_MC_DYNAMIC_STATE Changes
14	31	Min-State Save Area Size Change

2.2 Specification Clarifications

No.	Page	SPECIFICATION CLARIFICATIONS
1	34	Clarification of ptc.g release semantics
2	34	Clarification of PAL_MC_ERROR_INFO reporting of uncacheable transactions
3	35	Clarification of behavior when ptc.g overlaps a translation register
4	35	INT3 Clarifications
5	35	Test feature instruction clarifications
6	35	Clarification of performance counter behavior under halt states
7	37	PMI Clarifications
8	38	PAL_MC_ERROR_INJECT Clarifications
9	39	Min-state Save Area Clarifications
10	39	Semaphore Code Corrections

2.3 Documentation Changes

No.	Page	DOCUMENTATION CHANGES
1	41	Revision 2.2 Documentation Changes

3 Specification Changes

1. Illegal VAC/VDC combinations and IIPA requirements

1. In Volume 2, Part I, Chapter 11, add a new Section 11.7.4.3 “Virtualization Optimization Combinations”. In the new Section 11.7.4.3, add the following description and table to indicate the allowed *vac* and *vdc* combinations:

11.7.4.3 Virtualization Optimization Combinations

Table 11-35 describes the supported combinations of virtualization accelerations and disables.

Table 11-35. Supported Virtualization Optimization Combinations

	d_vmsw	d_extint	d_ibr_dbr	d_pmc	d_to_pmd	d_itm	d_psr_i
a_int	o[a]	x[b]	o	o	o	o	x
a_from_int_cr	o	o	o	o	o	o	o
a_to_int_cr	o	o	o	o	o	o	o
a_from_psr	o	o	o	o	o	o	x
a_from_cpuid	o	o	o	o	o	o	o
a_cover	o	o	o	o	o	o	o
a_bsw	o	o	o	o	o	o	x

Notes:

a. “o” indicates the corresponding virtualization acceleration and disable can be enabled together.
b. “x” indicates the corresponding virtualization acceleration and disable cannot be enabled together.

2. In Volume 2, Part I, Chapter 11, Section 11.7.4.2.7, "Disable PSR Interrupt-bit Virtualization", replace the note:

   Note: This field overrides the a_int Virtualization Acceleration Control (*vac*) described in Section 11.7.4.1.1, "Virtual External Interrupt Optimization" on page 2:323. If this control is enabled (set to 1), the a_int Virtualization Acceleration Control (*vac*) is ignored.

   with:

   Note: This field cannot be enabled together with a_int, a_from_psr or a_bsw virtualization accelerations. If this control is enabled together with any one of the described accelerations, an error will be returned during PAL_VP_CREATE and PAL_VP_REGISTER. See Section 11.7.4.3 for details.

3. In Volume 2, Part I, Chapter 11, Section 11.7.4.2.2 "Disable External Interrupt Control Register Virtualization”, replace the note:

   Note: This field overrides the a_int Virtualization Acceleration Control (*vac*) described in Section 11.7.4.1.1, "Virtual External Interrupt Optimization" on page 2:323. If this control is enabled (set to 1), the a_int Virtualization Acceleration Control (*vac*) is ignored.

   with:

   Note: This field cannot be enabled together with the a_int virtualization acceleration control (*vac*) described in Section 11.7.4.1.1. If this control is enabled together with the a_int control, an error will be returned during PAL_VP_CREATE and PAL_VP_REGISTER. See Section 11.7.4.3 for details.

4. In Volume 2, Part I, Chapter 11, Section 11.10, “PAL Procedures”, PAL_VP_CREATE page. In the Description section, after the second paragraph, last sentence:

   The *vac*, *vdc* and *virt_env_vaddr* parameters in the VPD must already be initialized before calling this procedure.

   Add the following sentence:

   Invalid argument is returned on unsupported *vac*/*vdc* combinations. See Section 11.7.4.3 for details.

5. In Volume 2, Part I, Chapter 11, Section 11.10, “PAL Procedures”, PAL_VP_REGISTER page. In the Description section, add the following paragraph before the last paragraph:

   PAL_VP_REGISTER returns invalid argument on unsupported virtualization optimization combinations in *vpd*. See Section 11.7.4.3, “Virtualization Optimization Combinations” for details.

6. In Volume 2, Part I, Chapter 11, Section 11.7.4.1.1 "Virtual External Interrupt Optimization", add the following note to the end of the section:

   Note: This field cannot be enabled together with d_extint or d_psr_i virtualization disables. If this control is enabled together with any one of described disables, an error will be returned during PAL_VP_CREATE and PAL_VP_REGISTER. See Section 11.7.4.3, “Virtualization Optimization Combinations” for details.

7. In Volume 2, Part I, Chapter 11, Section 11.7.4.1.4 "MOV-from-PSR Optimization", add the following note to the end of the section:

   Note: This field cannot be enabled together with the d_psr_i virtualization disable control (*vdc*) described in Section 11.7.4.2.7, “Disable PSR Interrupt-bit Virtualization”. If this control is enabled together with the d_psr_i control, an error will be returned during PAL_VP_CREATE and PAL_VP_REGISTER. See Section 11.7.4.3, “Virtualization Optimization Combinations” for details.

8. In Volume 2, Part I, Chapter 11, Section 11.7.4.1.7 "Bank Switch Optimization", add the following note to the end of the section:

   Note: This field cannot be enabled together with the d_psr_i virtualization disable control (*vdc*) described in Section 11.7.4.2.7. If this control is enabled together with the d_psr_i control, an error will be returned during PAL_VP_CREATE and PAL_VP_REGISTER. See Section 11.7.4.3, “Virtualization Optimization Combinations” for details.

2. Resouce Utilization Counter

The existing Interval Time Counter application register is clocked at a constant rate, independent of logical processor and virtual processor context switches on a processor core.

The new Resource Utilization Counter application register is clocked like the ITC, but is provided per logical or virtual processor and provides an estimate of the portion of resources used by a logical or virtual processor with respect to all resources provided by the underlying physical processor.

1. Add AR.ruc to Volume 1, Part I, Section 3.1.1, “Reserved and Ignored Registers and Fields”. In Figure 3-1, add RUC, AR 45 directly below and directly next to ITC, AR 44.

2. In Volume I, Part I, Section 3.1.8, “Application Registers”, Table 3-3, add the RUC entry as shown:

Register	Name	Description	Execution Unit Type
AR 44	ITC	Interval Time Counter	
AR 45	RUC	Resource Utilization Counter	
AR 46 - AR 47		Reserved	

   a. Add a new Resource Utilization Counter section just after Section 3.1.8.10, "Interval Time Counter":

3.1.8.11 Resource Utilization Counter (RUC - AR 45)

The Resource Utilization Counter (RUC) is a 64-bit register which provides an estimate of the portion of resources used by a logical or virtual processor with respect to all resources provided by the underlying physical processor.

In a given time interval, the difference in the RUC values for all of the logical or virtual processors on a given physical processor add up to approximately the difference seen in the ITC on that physical processor for that same interval. (See Vol 2, Section 11.7 for details on virtual processors.)

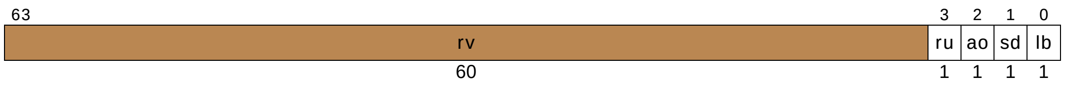
A sequence of reads of the RUC is guaranteed to return ever-increasing values (except for the case of the counter wrapping back to 0) corresponding to the program order of the reads.

System software can secure the resource utilization counter from non-privileged access. When secured, a read of the RUC at any privilege level other than the most privileged causes a Privileged Register fault.

3. Add CPUID[4] bit for non-privileged discovery of the Resource Utilization Counter

   a. In Volume 1, Part I, section 3.1.11, "Processor Identification Registers", Figure 3-12, add a new *ru* bit as shown:

Figure 3-12. CPUID Register 4 – General Features/Capability Bits



   b. In Volume 1, Part I, Section 3.1.11, “Processor Identification Registers”, add the *ru* entry to Table 3-8 as shown:

Table 3-8. CPUID Register 4 Fields

Field	Bits	Description
ru	3	Processor implements the Resource Utilization Counter (AR 45).
rv	63:4	Reserved.

4. In Volume 1, Part I, Section 6.2, “IA-32 Application Register State Model”, add AR.ruc to Fig 6-3 and Table 6-1.

   a. In Figure 6-3, add RUC, AR 45 directly below and directly next to ITC, AR 44.

   b. In Table 6-1, add a new row, just under the row for ITC, with this information:

RUC		Unmodified	64	RUC continues to count while in IA-32 execution mode

5. Describe serialization for RUC in Volume 2, Part I, Section 3.2.2 “Data Serialization”.
   a. In the second paragraph, change:
      ... RR, PKR, DTR, ...
      to:
      ... RR, PKR, RUC, DTR, ...
6. Add AR.ruc to Volume 2, Part I, Section 3.3.1 “System State Overview”.
   a. In the bullet list, just after the bullet for "Interval Timer Facilities", add a new bullet:
      - Resource Utilization Facility - A 64-bit resource utilization counter is provided for privileged and non-privileged use. This counts the number of Interval Timer cycles consumed by this logical processor. See Section 3.1.8.11, “Resource Utilization Counter” on page 1:29.
   b. Add AR.ruc to Figure 3-1:
      Add RUC, AR 45 directly below and directly next to ITC, AR 44.
7. Update the description of PSR.si to cover RUC.

   In Volume 2, Part I, Section 3.3.2 “Processor Status Register (PSR)”, Table 3-2, change this part of the description of the PSR.si bit:

   When 1, the Interval Time Counter (ITC) register is readable only by privileged code; non-privileged reads result in a Privileged Register fault. When 0, ITC is readable at any privilege level.

   to:

   When 1, the Interval Time Counter (ITC) register and the Resource Utilization Counter (RUC) are readable only by privileged code; non-privileged reads result in a Privileged Register fault. When 0, ITC and RUC are readable at any privilege level.
8. Add AR.ruc to Volume 2, Part I, Section 3.3.4 “Global Control Registers”.
   a. Add a new section, just after 3.3.4.2 "Interval Time Counter and Match Register":

3.3.4.3 Resource Utilization Counter (AR 45)

The Resource Utilization Counter (RUC) is a 64-bit counter that counts up at a fixed relationship to the input clock to the processor, when the processor is active. Processors may be inactive due to hardware multi-threading. Virtual processors may be inactive when not scheduled to run by the VMM. (See Vol 2, section 11.7 for details on virtual processors.)

The RUC is clocked such that, in a given time interval, the difference in the RUC values for all of the logical or virtual processors on a given physical processor add up to approximately the difference seen in the ITC on that physical processor for that same interval.

A sequence of reads of the RUC is guaranteed to return ever-increasing values (except for the case of the counter wrapping back to 0) corresponding to the program order of the reads. Applications can directly sample the RUC for active-running-time calculations.

A 64-bit overflow condition can occur without notification. The RUC can be read at any privilege level if PSR.si is zero. The timer can be secured from non-privileged access by setting PSR.si to one. When secured, a read of the RUC by non-privileged code results in a Privileged Register fault. Writes to the RUC can only be performed at privilege level 0; otherwise, a Privileged Register fault is raised.

Modification of the RUC is not necessarily serialized with respect to instruction execution. Software can issue a data serialization operation to ensure the RUC updates are observed by a given point in program execution. Software must accept a level of sampling error when reading the resource utilization counter due to various machine stall conditions, interruptions, bus contention effects, etc. Please see the processor-specific documentation for further information on the level of sampling error of the Itanium processor.

RUC should only be written by Virtual Machine Monitors; other Operating Systems should not write to RUC, but should only read it.

9. Update Volume 2, Part I, Section 3.4, “Processor Virtualization”, Table 3-10. Change the last two rows to the following:

Class	Virtualized Instructions

Reading AR[ITC] with PSR.si==1 takes (virtualized at all privilege levels)	`mov from ar.itc`
	`mov from ar.ruc`

Instructions which write privileged registers	`mov to itc`
	`mov to ar.ruc`

10. Update Volume 2, Part I, Section 11.3.2, "PALE_CHECK Exit State".

   a. In the bullet for ARs, change the following:

      The contents of all application registers are unchanged from the time of the MCA, except the RSE control register (RSC), the RSE backing store pointer (BSP), and the ITC counter.

   to:

      The contents of all application registers are unchanged from the time of the MCA, except the RSE control register (RSC), the RSE backing store pointer (BSP), and the ITC and RUC counters.

   b. In that same AR bullet, change the following:

      The ITC register will not be directly modified by PAL, but will continue to count during the execution of the MCA handler.

   to:

      The ITC register will not be directly modified by PAL, but will continue to count during the execution of the MCA handler. The RUC register will not be directly modified by PAL, but will Continue to count during the execution of the MCA handler while the processor is active.

11. Update Volume 2, Part I, Section 11.4.2, "PALE_INIT Exit State".

   a. In the bullet for ARs, change the following:

      The contents of all application registers are unchanged from the time of the INIT, except the RSE control register (RSC), the RSE backing store pointer (BSP), and the ITC counter.

   to:

      The contents of all application registers are unchanged from the time of the INIT, except the RSE control register (RSC), the RSE backing store pointer (BSP), and the ITC and RUC counters.

b. In that same AR bullet, change the following:

   The ITC register will not be directly modified by PAL, but will continue to count during the execution of the INIT handler.

   to:

   The ITC register will not be directly modified by PAL, but will continue to count during the execution of the INIT handler. The RUC register will not be directly modified by PAL, but will continue to count during the execution of the INIT handler while the processor is active.

12. Update Volume 2, Part I, Section 11.5.2, "PALE_PMI Exit State".

   a. In the bullet for ARs, change the following:

      The contents of all application registers are unchanged from the time of the interruption, except the RSE control register (RSC) and the ITC counter.

      to:

      The contents of all application registers are unchanged from the time of the interruption, except the RSE control register (RSC) and the ITC and RUC counters.

   b. In that same AR bullet, change the following:

      The ITC register will not be directly modified by PAL, but will continue to count during the execution of the PMI handler.

      to:

      The ITC register will not be directly modified by PAL, but will continue to count during the execution of the PMI handler. The RUC register will not be directly modified by PAL, but will continue to count during the execution of the PMI handler while the processor is active.

13. Update Volume 2, Part I, Section 11.7.3.1, "PAL Virtualization Intercept Handoff State".

   a. In the bullet for ARs, change the following:

      The contents of all application registers are preserved from the time of the interruption, except the ITC counter. The ITC register will not be directly modified by PAL, but will continue to count during the execution of the virtualization intercept handler.

      to:

      The contents of all application registers are preserved from the time of the interruption, except the ITC and RUC counters. The ITC register will not be directly modified by PAL, but will continue to count during the execution of the virtualization intercept handler. The RUC register will not be directly modified by PAL, but will continue to count during the execution of the virtualization intercept handler while the processor is active.

14. Update Volume 2, Part I, Section 11.10.2.2.7, "Application Registers". Add a new row to Table 11-48, just below the ITC row, with this information:

Register	Description	Class
RUC	Resource Utilization Counter	unchanged[c]

c. No PAL procedure writes to the RUC. The value at exit is the value at entry plus the number of cycles provided to the processor during the procedure call.

15. Update Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”, PAL_ENTER_IA_32_ENV page.

    a. Add a new row to the bottom of Table 11-67:

Intel® Itanium® Register	IA-32 State	Description
RUC	--	RUC continues to count while in IA-32 execution mode

    b. Add a new row to Table 11-71 just below the row for ITC:

Intel® Itanium® Register	IA-32 State	Description
RUC	--	Final value of RUC

16. Add a new PAL_VP_INFO procedure for privileged discovery of the Resource Utilization Counter.

    a. Add a new PAL procedure to Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”, just before PAL_VP_INIT_ENV:

PAL_VP_INFO – PAL Virtual Processor Information (50)

Purpose: Returns information about virtual processor features.

Calling Conv: Static

Mode: Physical

Buffer: Not dependent

Arguments:

Argument	Description
index	Index of PAL_VP_INFO within the list of PAL procedures.
feature_set	Feature set information is being requested for.
vp_buffer	An address to an 8-byte aligned memory buffer (if used).
Reserved	0.

Returns:

Return Value	Description
status	Return status of the PAL_VP_INFO procedure.
vp_info	Information about the virtual processor..
vmm_id	Unique identifier for the VMM.
Reserved	0

Status:

Status Value	Description
0	Call completed without error
-1	Unimplemented procedure
-2	Invalid argument
-3	Call completed with error
-8	Specified *feature_set* is not implemented

Description: The PAL_VP_INFO procedure call is used to describe virtual processor features.

The *feature_set* input argument for PAL_VP_INFO describes which virtual-processor *feature_set* information is being requested, and is composed of two fields as shown:

31 … 0
index

63 … 48	47 … 32
vmm_id	index

A *vmm_id* of 0 indicates architected feature sets, while others are implementation-specific feature sets. Implementation-specific feature sets are described in VMM-specific documentation.

This procedure will return a -8 if an unsupported *feature_set* argument is passed as an input. The return status is used by the caller to know which feature sets are currently supported on a particular VMM. This procedure always returns unimplemented (-1) when called on physical processors.

For each valid *feature_set*, this procedure returns information about the virtual processor in *vp_info*. Additional information may be returned in the memory buffer pointed to by *vp_buffer*, as needed. Details, for a given implementation-specific *feature_set*, of whether information is returned in the buffer, the size of the buffer, and the representation of this information in the buffer and in *vp_info* are described in VMM-specific documentation.

Architected *feature_set* 0 (*vmm_id* 0, *index* 0) is defined and required to be implemented (if this procedure is implemented), but there are no architected features defined in it yet, and so all bits in *vp_info* are reserved for architected *feature_set* 0.

Other architected feature sets (*vmm_id* 0, *index*>0) are undefined, and return -8 (Specified *feature_set* is not implemented). SW can call PAL_VP_INFO with a *feature_set* argument of 0 to get the *vmm_id*, although *vmm_id* is also returned for any other implemented feature sets as well. For *feature_set* 0, the *vp_buffer* arg is ignored.

b. Add PAL_VP_INFO to Volume 2, Part I, Section 11.10.1, "PAL Procedure Summary", Table 11-42.

Procedure	Idx	Class	Conv.	Mode	Buffer	Description
PAL_VP_INFO	50	Opt.	Static	Phys.	No.	Returns information about virtual processor features.

17. In Volume 3, Part I, Section 2.2, "Instruction Descriptions", update the "mov ar" instruction page pseudo-code.

a. In the Operation section, change the following "from form" code:

```
if (ar3 == ITC && PSR.si && PSR.cpl != 0)
      privileged_register_fault();

if (ar3 == ITC && PSR.si && PSR.vm == 1)
      virtualization_fault();"
```

to:

```
if ((ar3 == ITC || ar3 == RUC) && PSR.si && PSR.cpl != 0)
      privileged_register_fault();

if ((ar3 == ITC || ar3 == RUC) && PSR.si && PSR.vm == 1)
      virtualization_fault();"
```

b. In the to_form code, change the following "to form" code:

```
if ((is_kernel_reg(ar3) || ar3 == ITC) && (PSR.cpl != 0))
      privileged_register_fault();

if (ar3 == ITC && PSR.vm == 1)
      virtualization_fault();
```

to:

```
if ((is_kernel_reg(ar3) || ar3 == ITC || ar3 == RUC) && (PSR.cpl != 0))
      privileged_register_fault();

   if ((ar3 == ITC || ar3 == RUC) && PSR.vm == 1)
      virtualization_fault();
```

c. At the beginning of the Operation section, change the following code:

```
if (is_reserved_reg(tmp_type, ar3))
      illegal_operation_fault();
```

to:

```
if (!instruction_implemented(MOV_AR_RUC))
      illegal_operation_fault();

if (is_reserved_reg(tmp_type, ar3))
      illegal_operation_fault();
```

18. In Volume 3, Part I, Section 5.3.2, "RAW Dependency Table", add AR.ruc to the resource dependency tables.

a. In Table 5-2, add a row just under the row for AR[RSC] with this information:

Resource Name	Writers	Readers	Semantics of Dependency
AR[RUC]	mov-to-AR-RUC	br.ia, mov-from-AR-RUC	impliedF

b. In Table 5-2, in the row for PSR.cpl, add mov-from-AR-RUC and mov-to-AR-RUC to the readers in both of the sub-rows as shown:

Resource Name	Writers	Readers	Semantics of Dependency
PSR.cpl	epc, br.ret	**priv-ops**, br.call, brl.call, epc, **mov-from-AR-ITC**, **mov-to-AR-ITC**, **mov-to-AR-RSC**, **mov-to-AR-K**, **mov-from-IND-PMD**, **probe-all**, **mem-readers**, **mem-writers**, **lfetch-all**, **mov-from-AR-RUC, mov-to-AR-RUC**	implied
	rfi	**priv-ops**, br.call, brl.call, epc, **mov-from-AR-ITC**, **mov-to-AR-ITC**, **mov-to-AR-RSC**, **mov-to-AR-K**, **mov-from-IND-PMD**, **probe-all**, **mem-readers**, **mem-writers**, **lfetch-all**, mov-from-AR-RUC, mov-to-AR-RUC	impliedF

c. In Table 5-2, update the PSR.si row, as shown:

Resource Name	Writers	Readers	Semantics of Dependency
PSR.si	**sys-mask-writers-partial**[7], **mov-to-PSR-l**	**mov-from-PSR**	impliedF
		mov-from-AR-ITC, **mov-from-AR-RUC,**	data
	rfi	**mov-from-AR-ITC**, **mov-from-AR-RUC, mov-from-PSR**	impliedF

d. In Table 5-2, update the row for PSR.vm as shown:

Resource Name	Writers	Readers	Semantics of Dependency
PSR.vm	vmsw	**mem-readers**, **mem-writers**, **mov-from-AR-ITC**, **mov-from-IND-CPUID**, **mov-to-AR-ITC**, **priv-ops**\vmsw, cover, thash, ttag, **mov-from-AR-RUC**, mov-to-AR-RUC	implied
	rfi	**mem-readers**, **mem-writers**, **mov-from-AR-ITC**, **mov-from-IND-CPUID**, **mov-to-AR-ITC**, **priv-ops**\vmsw, cover, thash, ttag, mov-from-AR-RUC, mov-to-AR-RUC	impliedF

e. In Volume 2, Part I, Section 5.3.3, "WAW Dependency Table", Table 5-3, add a row just under the row for AR[RSC] with this information:

Resource Name	Writers	Semantics of Dependency
AR[RUC]	mov-to-AR-RUC	impliedF

f. In Volume 3, Part I, Section 5.4 "Support Tables", Table 5-5, add a row just under the row for mov-from-AR-RSC with this information:

Class	Events/Instructions
mov-from-AR-RUC	mov-from-AR-M[Field(ar3) == RUC]

g. In Volume 3, Part I, Section 5.4 "Support Tables", Table 5-5, add a row just under the row for mov-to-AR-RSC with this information:

Class	Events/Instructions
mov-to-AR-RUC	mov-to-AR-M[Field(ar3) == RUC]

19. In Volume 2, Part II, add a new section just after Section 10.5.5, "Interval Timer Usage Example":

3.3.5 10.5.6 Resource Utilization Counter Usage Example

The Itanium architecture provides a 64-bit counter to provide information on how many execution cycles a given logical processor is getting. It is similar to the Interval Timer (ITC, AR 44), except that it is clocked only when the logical processor is active. Optimizations such as hardware multi-threading and processor virtualization may cause a logical processor to sometimes be inactive. The Resource Utilization Counter allows for better cycle accounting for logical processors, given these types of optimizations.

RUC should only be written by Virtual Machine Monitors; other Operating Systems should not write to RUC, but should only read it.

3. PAL_VP_INIT and VPD.vpr changes

1. PAL_VP_INIT_ENV currently freezes performance registers by clearing PMC[0].fr when fr_pmc is enabled. The following change removes the race condition that can occur when a counter overflow happens just before the write to PMC[0].fr, causing PAL to overwrite the overflow bit, and losing overflow information.

a. In Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”, PAL_VP_INIT_ENV procedure, Table 11-110, change the *fr_pmc* description from:

fr_pmc	1	If 1, performance counters are frozen on all IVA-based interruptions when virtual processors are running. If 0, the performance counters will not be frozen on IVA-based interruptions when virtual processors are running.

to:

fr_pmc	1	If 1, for virtualization intercepts the performance counters are disabled by setting PSR.up and pp to 0, see Section 11.7.3.1 for details on PSR settings at virtualization intercepts; for all other IVA-based interruptions PSR.pp and up are set according to Interruption State column described in Processor Status Field table described in Vol 2 Table 3-2. If 0, PSR.pp and up are set according to Interruption State column described in Processor Status Field table described in Vol 2 Table 3-2.

b. Volume 2, Part I, Section 11.7.3.1, "PAL Virtualization Handoff State", PSR description. Change the following bullet:

- PSR: PSR fields are set according to the "Interruption State" column in Table 3-2, "Processor Status Register Fields" on page 2:21.

to:

- PSR: PSR fields are set according to the "Interruption State" column in Table 3-2, "Processor Status Register Fields" on page 2:21. PSR.up and pp are set to 0 when *fr_pmc* field in *config_options* parameter during PAL_VP_INIT_ENV is 1.

2. Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”, PAL_VP_INIT_ENV page, Table 11-110, change the be *config_options* description from:

be	2	Big-endian – Indicates the endian setting of the VMM. If 1, the values in the VPD are stored in big-endian format and the PAL services calls are made with PSR.be bit equals to 1. If 0, the values in the VPD are stored in little-endian format and the PAL services calls are made with PSR.be bit equals to 0.

to:

be	2	Big-endian – Indicates the endian setting of the VMM. If 1, the values in the VPD are stored in big-endian format and the PAL services calls are made with PSR.be bit equals to 1. If 0, the values in the VPD are stored in little-endian format and the PAL services calls are made with PSR.be bit equals to 0. The VMM must match DCR.be with the value set in this field when the IVA control register on the logical processor is set to point to the per-virtual-processor host IVT. See Section 11.17.2 "Interruption Handling in a Virtual Environment" and Table 11-17 "IVA Settings after PAL Virtualization-related Procedures and Services" for details on per-virtual-processor host IVT".

3. Volume 2, Part I, Section 11.7.1,"Virtual Processor Descriptor", Table 11-14, Update the *vpr* row from:

vpr	1	1432	Virtual Predicate Registers – Represents the Predicate Registers of the virtual processor. The bit positions in vpr correspond to predicate registers in the same manner as with the mov predicates instruction.	Architectural State [always]

to:

vpr	1	1432	Virtual Predicate Registers – Represents the Predicate Registers of the virtual processor. The bit positions in vpr correspond to predicate registers in the same manner as with the mov predicates instruction. The contents in this field are undefined except at virtualization intercept handoff. The VMM can not rely on the contents in this field to be preserved when the virtual processor is running.	Architectural State [always]

4. New PAL_VPS_RESUME_HANDLER to indicate RSE Current Frame Load Enable setting at the target instruction

This change adds a parameter to PAL_VPS_RESUME_HANDLER to allow a virtual machine monitor to specify the register stack engine Current Frame Load Enable setting at the target guest handler.

1. Volume 2, Part I, Section 11.11.12, “PAL Virtualization Service Specifications”, PAL_VPS_RESUME_HANDLER page.

    a. Change the description of GR26 from:

        Virtualization Acceleration Control (*vac*) field from the VPD specified in GR25

    to:

        Virtualization Acceleration Control (*vac*) field from the VPD specified in GR25 and CFLE setting at the target instruction.

    b. In the second paragraph, change the following paragraph from:

        The VMM specifies the BR0 of the virtual processor in GR24, the 64-bit virtual pointer to the VPD in GR25 and the *vac* field of the VPD in GR26.

Behavior is undefined if the *vac* in GR26 does not match the *vac* field in the VPD argument specified in GR25.

to:

GR24 specifies the BR0 of the virtual processor; GR25 specifies the 64-bit virtual pointer to the VPD; GR26 specifies the *vac* field of the VPD in GR26; bit 63 of GR26 specifies the value of CFLE setting at the target instruction. Behavior is undefined if the *vac* in GR26 does not match the *vac* field in the VPD argument specified in GR25.

2. In Volume 3, Part I, Section 2.2, "Instruction Descriptions", "tf" page:
   a. In the Description section, remove the following text:

      Implementation of PSR.vm is optional. If it is implemented but the instruction is disabled, this instruction takes Virtualization fault when executed with PSR.vm equals to 1.
   b. In the Operation section, remove both instances of the following lines:
```
if (PSR.vm == 1 && vm_tf_disabled())
virtualization_fault();
```
   c. In the Interruptions section, remove "Virtualization fault" from the list of interruptions.

5. PAL_VP_INIT_ENV Implementation-specific Configuration Option

This change defines an implementation-specific configuration bit for PAL_VP_INIT_ENV. A separate update to the "Dual-Core Update to the Itanium 2 Processor Reference Manual" will define this implementation-specific bit to optimize performance for virtual machine monitors using data translation cache for pages containing virtualized instructions.

1. Volume 2, Part 1, Section 11.10.3, "PAL Procedures Specifications", PAL_VP_INIT_ENV page. Add the *impl* bit to Table 11-110:

Field	Bit	Description
impl	63	Implementation-specific configuration option. This field is ignored if not implemented. Please refer to processor-specific documentation for details..

2. A future revision of the "Dual-Core Update to the Itanium 2 Processor Reference Manual" will include the following change:

Add a section to describe PAL_VP_INIT_ENV:

Table 11-62. PAL_VP_INIT_ENV Implementation-specific Behavior

Field	Bit	Description
hint_dtc	63	If 1, this hint indicates the VMM is using data translation cache for pages containing virtualized instructions. Instruction TLB misses will happen during virtualized instruction execution if the corresponding data translation does not exist in the TLB hierarchy

6. Increase in minimum number of virtual address bits

This change increases the minimum number of implemented virtual address bits from 51 to 54. Note that Itanium 2 processors and Dual Core Itanium 2 processors already support the 54 bit virtual address minimum.

1. Volume 2, Part 1, Section 4.3.2, "Unimplemented Virtual Address Bits" change the first paragraph:

> ... all processor models implement at least 51 virtual address bits; i.e., the smallest IMPL_VA_MSB is 50.

to:

> ... all processor models implement at least 54 virtual address bits; i.e., the smallest IMPL_VA_MSB is 53.

2. Change the second paragraph of Volume 2, Part 1, Section 4.3.2, “Unimplemented Virtual Address Bits” from:

> If the PSR.vm bit is implemented, at least 52 virtual address bits must be implemented.

to:

> If the PSR.vm bit is implemented, at least 55 virtual address bits must be implemented.

7. PAL_MC_ERROR_INFO health indicator

This change defines PAL_MC_ERROR_INFO cache_check, tlb_check, and uarch_check fields to allow hardware status tracking to be reported for processor structures. A new PAL_MC_HW_TRACKING procedure allows software to determine which processor structures provide hardware status tracking.

1. In Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”, PAL_MC_ERROR_INFO page, Table 11-82, add a *hlth* field row and a table note to the cache_check definiton:

Field	Bits	Description
hlth	31:30	Health indicator. This field will report if the cache type and level reporting this error supports hardware status tracking and the current status of this cache. 00 - No hardware status tracking is provided for the cache type and level reporting this event. 01 - Status tracking is provided for this cache type and level and the current status is normal status.(a) 10 - Status tracking is provided for the cache type and level and the current status is cautionary. (a) When a cache reports a cautionary status the "hardware damage" bit of the PSP (See Section 11.3.2.1, “Processor State Parameter (GR18)”) will be set as well. 11 - Reserved
rsvd	31:24	Reserved

(a) Hardware is tracking the operating status of the structure type and level reporting the error. The hardware reports a "normal" status when the number of entries within a structure reporting repeated corrections is at or below a pre-defined threshold. A "cautionary" status is reported when the number of affected entries exceeds a pre-defined threshold.

2. In Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”, PAL_MC_ERROR_INFO page, Table 11-83, add a new *hlth* field row and a table note to the tlb_check definition:

Field	Bits	Description
hlth	31:30	Health indicator. This field will report if the tlb type and level reporting this error supports hardware status tracking and the current status of this tlb. 00 - No hardware status tracking is provided for the tlb type and level reporting this event. 01 - Status tracking is provided for this tlb type and level and the current status is normal.(a) 10 - Status tracking is provided for the tlb type and level and the current status is cautionary(a) When a tlb reports a cautionary status the "hardware damage" bit of the PSP (See Section 11.3.2.1, “Processor State Parameter (GR18)”) will be set as well. 11 - Reserved
rsvd	31:24	Reserved

(a) Hardware is tracking the operating status of the structure type and level reporting the error. The hardware reports a "normal" status when the number of entries within a structure reporting repeated corrections is at or below a pre-defined threshold. A "cautionary" status is reported when the number of affected entries exceeds a pre-defined threshold.

3. Add a new PAL procedure called PAL_MC_HW_TRACKING

   a. Add PAL_MC_HW_TRACKING to Volume 2, Part I, Section 11.10.1, “PAL Procedure Summary”, Table 11-38:

Procedure	Idx	Class	Conv.	Mode	Buffe r	Description
PAL_MC_HW_TRACKING	51	Opt.	Static	Both	Yes	Query which hardware structures are performing hardware status tracking.

   b. In Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”, add a new PAL_MC_HW_TRACKING page after the PAL_MC_EXPECTED page:

PAL_MC_HW_TRACKING - Query which hardware structures are performing hardware status tracking (51)

Purpose: Provide a way to query which hardware structures are performing hardware status tracking for corrected machine check events.

Calling Conv: Static

Mode: Physical and Virtual

Buffer: Not dependent

Arguments:

Argument	Description
index	Index of PAL_MC_HW_TRACKING within the list of PAL procedures.
Reserved	0
Reserved	0
Reserved	0

Returns:

Return Value	Description
status	Return status of the PAL_MC_HW_TRACKING procedure.
hw_track	64-bit vector denoting which hardware structures are providing hardware status tracking. See Fig 11-100.
Reserved	0
Reserved	0

Status:

Status Value	Description
0	Call completed without error
-1	Unimplemented procedure
-2	Invalid argument
-3	Call completed with error

This procedure will return information about which hardware structures are providing hardware status tracking for corrected machine check events. This information is also returned in the error logs for corrected machine check events.

The layout of the tracked return value is showing in Fig 11-64.

Figure 11-64. Layout of *hw_track* return value

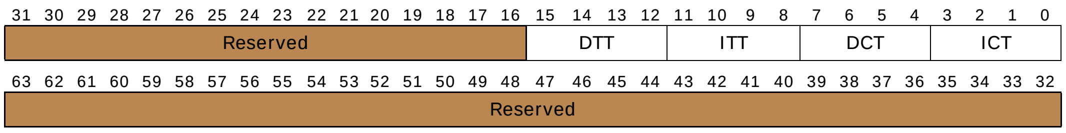


Table 11-100. *hw_track* Description

Field	Bits	Description
ICT	3:0	Instruction cache tracking. This is a 4-bit vector denoting which levels of instruction cache provide hardware tracking.
DCT	7:4	Data cache tracking. This is a 4-bit vector denoting which levels of the data/unified caches provide hardware tracking.
ITT	11:8	Instruction TLB tracking. This is a 4-bit vector denoting which levels of the instruction TLB provide hardware tracking.
DTT	15:12	Data TLB tracking. This is a 4-bit vector denoting which levels of data/unified TLB provide hardware tracking
rsvd	63:16	Reserved.

The convention for the levels in the *hw_track* field is such that the least significant bit in the field represents the lowest level of the structures hierarchy. For example bit 0 of the ICT field represents the first level instruction cache.

8. New implementation-specific bit fields for PAL_MC_ERROR_INJECT

This change defines implementation-specific bits for PAL_MC_ERROR_INJECT.

1. In Volume 2, Part I, Section 11.10.3, PAL_MC_ERROR_INJECT procedure, Figure 11-51 change bits 63:48 from Reserved to implementation specific in Figure 11-51 as shown:

Figure 11-51. *err_type_info*

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13	12 11 10 9 8	7 6	5 4 3	2 1 0
Reserved	struct_hier	err_struct	err_sev	err_inj	mode

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48	47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32
Impl_Spec	Reserved

2. In Volume 2, Part I, Section 11.10.3, PAL_MC_ERROR_INJECT procedure, Table 11-87, change bits 63:48 from Reserved to implementation specific as shown:

Field	Bits	Description
Reserved	47:16	Reserved
impl_spec	63:48	Processor specific error injection capabilities. Please refer to processor specific documentation for additional details.

9. MOV-to-PSR.L Reserved Field Checking

This change relaxes the architectural requirement for checking the reserved upper 32 bits on MOV-to-PSR.L, making this check implementation-specific.

Volume 3, Part I, Section 2.2, “Instruction Descriptions, “mov - Move Processor Status Register” page. In the Description, change the third paragraph from:

> For move to processor status register, GR r_2 is read, bits {31:0} copied into PSR{31:0} and bits{45:32} are ignored. All bits of GR r_2 corresponding to reserved fields of the PSR must be 0 or a Reserved Register/Field fault will result.

to:

> For move to processor status register, GR r_2 is read, bits {31:0} copied into PSR{31:0} and bits {63:32} are ignored. Bits {31:0} of GR r_2 corresponding to reserved fields of the PSR must be 0 or a Reserved Register/Field fault will result. An implementation may also raise Reserved Register/Field fault if bits {63:32} in GR r_2 corresponding to reserved fields of the PSR are non-zero.

10. Virtual Machine Disable

This change defines a mechanism to disable processor virtualization features.

1. 2. In Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”, PAL_PROC_GET_FEATURES page, Table 11-103, change bit 40 to the following:

Bit	Class	Control	Description
40	Opt.	Opt	Virtual Machine features implemented and enabled. When 1, PSR.vm is implemented and virtual machines features are not disabled. When 0 (*features_status*) and when the corresponding *features_avail* bit is 1, virtual machines features are implemented but are disabled. When both the *features_avail* and *features_status* bits are 0, virtual machine features are not implemented. If implemented and controllable, virtual machine features may be disabled by writing this bit to 0 with PAL_PROC_SET_FEATURES. However, virtual machine features cannot be re-enabled except via a reset; hence, if virtual machine features are disabled, this bit reads as 0 for both *features_status* and *features_control* (but still 1 for *features_avail*).

2. Volume 2, Part I, Section 3.4, "Processor Virtualization". Add the following paragraph before the last paragraph:

   Processors which support processor virtualization may provide an implementation-dependent mechanism to disable virtual machine features, see PAL_PROC_GET_FEATURES on page 2:429 for details.

3. Volume 3, Part I, Section 2.2, “Instruction Descriptions”, vmsw page. In the last sentence of the last paragraph:

   See Section 3.4, "Processor Virtualization" on page 2:40 and PAL_PROC_GET_FEATURES on page 2:433 for details.

   add a reference to PAL_PROC_SET_FEATURES:

   See Section 3.4, "Processor Virtualization" on page 2:40, PAL_PROC_GET_FEATURES and PAL_PROC_SET_FEATURES on page 2:429 and page 2:433 for details.

4. Volume 2, Part I, Section 11.8, "PAL Glossary", add the following two definitions:

Power-on

The reset event that occurs when the power input to the processor is applied and the reset input to the processor is asserted.

Reset

The reset event that occurs when the reset input to the processor is asserted.

5. Volume 2, Part I, Section 11.2.1, “PALE_RESET”. In the first sentence, change the following text:

   Upon receipt of a power-on reset event the processor begins executing code from...

   to:

   Upon receipt of a power-on/reset event the processor begins executing code from...

6. Volume 3, Part I, Chapter 3, Table 3-1, "Pseudo-code Functions.” Change the description of the function "implemented_vm" from:

   Returns TRUE if the processor implements the PSR.vm bit.

   to:

   Returns TRUE if the processor implements the PSR.vm bit (regardless of whether virtual machine features are enabled or disabled).

7. Volume 3, Part I, Chapter 3, Table 3-1 "Pseudo-code Functions", rename the function "vm_disabled" to "vmsw_disabled".

8. Volume 3, Part I, Chapter 3, Table 3-1 "Pseudo-code Function", add a new function "vm_disabled":

Function	Operation
vm_disabled	Returns TRUE if the processor implements the PSR.vm bit and virtual machine features are disabled. See Section 3.4, "Processor Virtualization" on page 2:40 in SDM and "PAL_PROC_GET_FEATURES - Get Processor Dependent Features (17)" on page 2:433 for details.

9. Volume 3, Part I, Chapter 2, VMSW I-page, change the line:

```
if (!(PSR.it == 1 && itlb_ar() == 7) || vm_disabled())
```

to:

```
if (!(PSR.it == 1 && itlb_ar() == 7) || vm_disabled() ||
vmsw_disabled())
```

10. Volume 3, Part I, Section 2.2, “Instruction Descriptions”, vmsw page. In the description section, change the last paragraph from:

    Implementation of PSR.vm is optional. If it is not implemented, this instruction takes Illegal Operation fault. If it is implemented but is disabled, this instruction takes Virtualization fault when executed at the most privileged level. See Section 3.4, "Processor Virtualization" on page 2:40 and PAL_PROC_GET_FEATURES on page 2:433 for details.

    to:

    Implementation of PSR.vm is optional. If it is not implemented, this instruction takes Illegal Operation fault. If it is implemented but either virtual machine features or the vmsw instruction are disabled, this instruction takes Virtualization fault when executed at the most privileged level. See Section 3.4, "Processor Virtualization" on page 2:40 and PAL_PROC_GET_FEATURES on page 2:433 for details.

11. Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”. In the PAL_VP_ENV_INFO page, add the following paragraph to the end of the section (just before Table 11-109):

    This procedure returns unimplemented procedure when virtual machine features are disabled. See Section 3.4, "Processor Virtualization" on page 2:40 and PAL_PROC_GET_FEATURES on page 2:433 for details.

12. Volume 2, Part I, Chapter 11, Section 11.10.3, “PAL Procedure Specifications”, PAL_VP_INIT_ENV page, add the following paragraph at the end of the Description:

    This procedure returns unimplemented procedure when virtual machine features are disabled. See Section 3.4, "Processor Virtualization" on page 2:40 and PAL_PROC_GET_FEATURES on page 2:433 for details.

13. Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications, PAL_PROC_GET_FEATURES procedure, Table 11-103, add this sentence to the end of the description for bit 54, "Enable the use of the vmsw instruction":

    This bit has no effect if virtual machine features are disabled (see bit 40).

14. Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”, PAL_PROC_GET_FEATURES procedure, add a sentence to the following existing paragraph on the PAL_PROC_GET_FEATURES page:

    For each valid *feature_set*, this procedure returns which processor features are implemented in the *features_avail* return argument, the current feature setting is in *feature_status* return argument, and the feature controllability in the feature_control return argument. Only the processor features which are implemented and controllable can be changed via PAL_PROC_SET_FEATURES.

    to:

    For each valid *feature_set*, this procedure returns which processor features are implemented in the *features_avail* return argument, the current feature setting is in *feature_status* return argument, and the feature controllability in the feature_control return argument. Only the processor features which are implemented and controllable can be changed via PAL_PROC_SET_FEATURES. Features for which *features_avail* are 0 (unimplemented features) also have *features_status* and *features_control* of 0.

11. Removal of *pal_proc_vector* argument from PAL_VP_SAVE and PAL_VP_RESTORE

This change simplifies PAL_VP_SAVE and PAL_VP_RESTORE implementations by removing the *pal_proc_vector* argument from these calls.

1. Volume 2, Part I, Section 11.10.3, "PAL Procedure Specifications", PAL_VP_RESTORE page:
   a. In the Argument section, change "pal_proc_vector" and corresponding description to "Reserved" and "0".
   b. In the Description section, remove the paragraph:

      The *pal_proc_vector* parameter for PAL_VP_RESTORE allows the VMM to control the PAL procedure implementation-specific state to be saved. Table 11-111 shows the format of *pal_proc_vector.* When a bit is set to 1 in the vector, the implementation-specific state for the corresponding PAL procedures will be restored by PAL_VP_RESTORE. When a bit is set to 0 in the vector, no implementation-specific state will be restored for the corresponding PAL procedures.
   c. Remove Table 11-111, "Format of *pal_proc_vector*".
2. Volume 2, Part I, Section 11.10.3, "PAL Procedure Specifications", PAL_VP_SAVE page:
   a. In the Argument section, change "pal_proc_vector" and corresponding description to "Reserved" and "0".
   b. In the Description section, remove the following paragraph:

      The *pal_proc_vector* parameter for PAL_VP_SAVE allows the VMM to control the PAL procedure implementation-specific state to be saved. Table 11-111 on page 2:463 shows the format of *pal_proc_vector*. When a bit is set to 1 in the vector, the implementation-specific state for the corresponding PAL procedures will be saved by PAL_VP_SAVE. When a bit is set to 0 in the vector, no implementation-specific state will be saved for the corresponding PAL procedures.

12. Variable Frequency Mode Additions to ACPI P-states

1. In Volume 2, Part I, Section 11.6.1, "Power/Performance States (P-states)", change the following text before Figure 11-22:

   PAL_GET_PSTATE: This procedure returns the performance index of the logical processor, relative to the highest available P-state P0 which has an index value of 100. For example, if the value returned by the procedure is 80, it indicates that the performance of the logical processor over the last time period was 20% lower than the P0 performance capability of the logical processor. The performance index is measured over the time interval since the last PAL_GET_PSTATE call.

   to:

   PAL_GET_PSTATE: This procedure returns the performance index of the logical processor, relative to the highest available P-state P0. A value of 100 in P0 represents the minimum processor performance in the P0 state. For example, if the value returned by the procedure is 80, this indicates that the performance of the logical processor over the last time period was 20% lower than the minimum P0 performance. For processors that support variable P-states, it is possible for a processor to report a number greater than 100, representing that the processor is running at a performance level greater than the minimum P0 performance. For example, if the value returned by the processor is 120, it indicates that the performance of the logical processor over the last time period was 20% higher than the minimum P0 performance. The performance index is measured over the time interval since the last PAL_GET_PSTATE call with a type operand of 1. If the processor supports variable P-state performance then the PAL_PROC_SET_FEATURE procedure can be used to enable or disable this feature.

2. Add the following text to Volume 2, Part I, Section 11.6.1, “Power/Performance States (P-states)”, just before section 11.6.1.1:

Some processors may support variable P-state performance where the frequency within a given P-state may vary to achieve the maximum performance for that P-state's power budget. The PAL_PROC_GET_FEATURES procedure on page 2:429 indicates if the processor supports variable P-state performance.

The performance index calculation is slightly different when a processor supports variable P-state performance, since the frequency within a P-state can vary. These frequencies for a given P-state are represented by an index value Fx,y. The value x is the P-state number and y represents a frequency point in the range from 0 to N. A value of 0 represents the minimum frequency index value for the given P-state. For example:

$F_{0,0}$ to $F_{0,N}$ - Frequency index values for the P0 state

$F_{1,0}$ to $F_{1,N}$ - Frequency index values for the P1 state

etc..

$F_{0,0}$ is the minimum frequency index for the P0 state and its value is 100. $F_{0,1}$ represents a higher frequency point for P0 and will have a value greater than 100. For example if $F_{0,1}$ frequency is 5% greater than $F_{0,0}$ it would have a value of 105.

The *performance_index* equation for P0 is calculated as follows:

($F_{0,0}$ * time spent in $F_{0,0}$) + ($F_{0,1}$ * time spent in $F_{0,1}$)+ .. ($F_{0,N}$ * time spent in $F_{0,N}$) / (Total Time spent in P0)

For example let's say the minimum frequency of P0 is 1GHz and the maximum frequency of P0 is 1.5GHz. If we are at 1GHz for a time period of 4, 1.25GHz for a time period of 16 and 1.5GHz for a time period of 20, the average performance index is: (100*4) + (125*16) + (150*20) / (5+15+20) = 135

The *performance_index* equation for other P-states can be calculated in a similar manner using their respective frequency index values.

The total *performance_index* equation for a processor with four P-states (P0, P1, P2, P3) would be:

($F_{0,0}$ * time spent in $F_{0,0}$) + ($F_{0,1}$ * time spent in $F_{0,1}$)+ .. ($F_{0,N}$ * time spent in F0,N)+

($F_{1,0}$ * time spent in $F_{1,0}$) + ($F_{1,1}$ * time spent in $F_{1,1}$)+ .. ($F_{1,N}$ * time spent in F1,N)+

($F_{2,0}$ * time spent in $F_{2,0}$) + ($F_{2,1}$ * time spent in $F_{2,1}$)+ .. ($F_{2,N}$ * time spent in F2,N)+

($F_{3,0}$ * time spent in $F_{3,0}$) + ($F_{3,1}$ * time spent in $F_{3,1}$)+ .. ($F_{3,N}$ * time spent in $F_{3,N}$)

/ (Total Time)

3. Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”, PAL_GET_PSTATES page.

   a. Change the *performance_index* return value definition from:

      Unsigned integer denoting the processor performance for the time duration since the last PAL_GET_PSTATE procedure call was made. The value returned is between 0 and 100, and is relative to the performance index of the highest available P-state."

      to:

      Unsigned integer denoting the processor performance for the time duration since the last PAL_GET_PSTATE procedure call was made. The value returned is relative to the performance index of the highest available P-state.

   b. Change the first paragraph of the Description section:

      This procedure returns the performance index of the processor over the time period between the previous and the current invocations of PAL_GET_PSTATE, and is relative to the highest available P-state. For processors that belong to a software-coordinated dependency domain or a hardware-independent dependency domain, the *performance_index* value returned will correspond to the target P-state requested by the most recent PAL_SET_PSTATE procedure call.

      to:

      This procedure returns the performance index of the processor over the time period between the previous and the current invocations of PAL_GET_PSTATE, and is relative to the highest available P-state, P0. A value of 100 represents the minimum processor performance in the P0 state. For processors that support variable P-state performance, it is possible for a processor to report a number greater than 100, representing that the processor is running at a performance level greater than the minimum P0 performance. The PAL procedure PAL_PROC_GET_FEATURES on page 2:429 indicates if the processor supports variable P-state performance.

      For processors that belong to a software-coordinated dependency domain or a hardware-independent dependency domain, the *performance_index* value returned will correspond to the target P-state requested by the most recent PAL_SET_PSTATE procedure call in cases where variable P-state performance is not supported. When variable P-states performance is supported, the *performance_index* may be higher than the target P-state requested. Please see Section 11.6.1 for more information about variable P-state performance.

   c. In the Description section, change the second paragraph after Table 11-73.

      If there was a thermal-throttling event or any hardware-initiated event, which affected the processor power/performance for the current time period and the accuracy of the *performance_index* value has been impacted by the event, then the procedure will return with status=1. The *performance_index* returned in this case will still have a value between 0 and 100.

      to:

      If there was a thermal-throttling event or any hardware-initiated event which affected the processor power/performance for the current time period and the accuracy of the *performance_index* value has been impacted by the event, then the procedure will return with status=1. The *performance_index* returned in this case will still have a value that falls within the range of possible *performance_index* values for this processor implementation. (i.e. 0 up to the highest variable p-state *performance_index* value)

4. In Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”, PAL_PROC_GET_FEATURES page, modify the bit 39 description as shown:

Bit	Class	Control	Description
39	Opt.	Req.	Variable P-state performance: A value of 1 indicates that the processor is optimizing performance for the given P-state power budget by dynamically varying the frequency, such that maximum performance is achieved for the power budget. A value of 0 indicates that P-states have no frequency variation or very small frequency variations for their given power budget.

13. PAL_MC_DYNAMIC_STATE Changes

1. Volume 2, Part I, Setction 11.10.1, “PAL Procedure Summary”, Table 11-38. Change the PAL_MC_DYNAMIC_STATE “Mode” value from “Phys.” to “Both”.
2. Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”. Update the PAL_MC_DYNAMIC page as follows:

PAL_MC_DYNAMIC_STATE – Returns Dynamic Processor State (24)

Purpose: Returns the Machine Check Dynamic Processor State.

Calling Conv: Static Registers Only

Mode: Physical and Virtual

Buffer: Not dependent

Arguments:

Arguments	Description
index	Index of PAL_MC_DYNAMIC_STATE within the list of PAL procedures.
info_type	Unsigned 64-bit value indicating the type of information to return
dy_buffer	64-bit pointer to a buffer aligned on an 8-byte boundary
Reserved	0

Returns:

Return Value	Description
status	Return status of the PAL_MC_DYNAMIC_STATE procedure.
max_size	Maximum size in bytes of the data that can be returned by this procedure for this processor family.
Reserved	0
Reserved	0

Status:

Status Value	Description
0	Call completed without error
-1	Unimplemented procedure
-2	Invalid argument
-3	Call completed with error

Description: The *info_type* input argument designates the type of information the procedure will return. When *info_type* is 0, the procedure returns the maximum size in bytes of processor dynamic state that can be returned for this processor family in the *max_size* return value.

When *info_type* is 1, the procedure will copy processor dynamic state into memory pointed to by the input argument *dy_buffer*. This copy will occur using the addressing attributes used to make the procedure call (physical or virtual) and the caller needs to ensure the *dy_buffer* input pointer matches this addressing attribute.

The amount of data returned can vary depending on the state of the machine when called and may not always return the maximum size for every call. The amount of data returned is provided in the processor state parameter field *dsize*. Please see Table 11-7 for more information on the processor state parameter. The caller of the procedure needs to ensure that the buffer is large enough to handle the *max_size* that is returned by this procedure.

The contents of the processor dynamic state is implementation dependent. Portions of this information may be cleared by the PAL_MC_CLEAR_LOG procedure. This procedure should be invoked before PAL_MC_CLEAR_LOG to ensure all the data is captured.

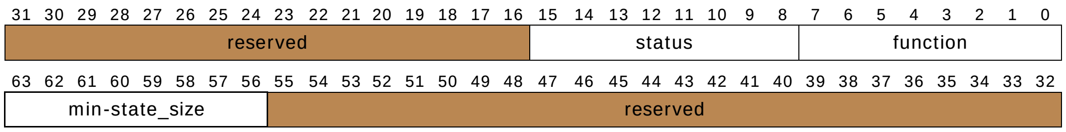
14. Min-State Save Area Size Change

1. In Volume 2, Part I, Section 11.2.2.1, “Definition of SALE_ENTRY State Parameter”, change the reset hand-off state to return a min-state save area size.

   Page 2:282 - Modify the existing SALE_ENTRY state parameter for reset and recovery check to pass the size of the min-state save area using some previously reserved fields

   a. Modify Figure 11-8 to add a new field called *min-state_size* as shown:

Figure 11-8. SALE_ENTRY State Parameter



   b. Add a bullet at the end of Section 11.2.2.1, after the sentence "For the case of RECOVERY CHECK, authentication of PAL_A and PAL_B should be completed before call to SAL_ENTRY:

      - *min-state_size* - An 8-bit field indicating the size in kilobytes (KB) of the min-state save area required for this implementation. A value of zero indicates a size of 4KB. A value greater than zero indicates the actual size in KB of the min-state save area required for this implementation. Values of 1-4 are reserved. For more information about the min-state save area, please refer to Section 11.3.2.3.

2. Volume 2, Part I, Section 11.3.2.3, “Processor Min-state Save Area Layout”.

   a. Change the first paragraph of Section 11.3.2.3 from:

      The processor min-state save area is 4KB in size and must be in an uncacheable region. The first 1KB of this area is architectural state needed by the PAL code to resume during MCA and INIT events (architected min-state save area + reserved). The remaining 3KB is a scratch buffer reserved exclusively for PAL use, therefore SAL and OS must not use this area. The layout of the processor min-state save area is shown in Figure 11-13.

   to:

      The processor min-state save area is minimally 4KB in size, but an implementation may require larger sizes. The reset hand-off state indicates if a size greater than 4KB is required and also provides the required size. Please refer to Section 11.2.2.1 for more information on the reset hand-off state. The required size is referred to as MIN_STATE_REQ. The min-state save area is required to be in an uncacheable region. The first 1KB of this area is architectural state needed by the PAL code to resume during MCA and INIT events (architected min-state save area + reserved). The remaining space in the buffer is a scratch space reserved exclusively for PAL use, therefore SAL and OS must not use this area. The layout of the processor min-state save area is shown in Figure 11-13.

   b. Figure 11-13 needs to be modified in two places to use the MIN_STATE_REQ variable:

      Change "Min-state save ptr + 4KB" to "Min-state save ptr + MIN_STATE_REQ" and change "3KB" to "MIN_STATE_REQ - 1KB".

c. Change the third paragraph of Section 11.3.2.3 from:

> The base address of the min-state save area must minimally be aligned on a 512-byte boundary, but larger alignments like 4 KB are fine."

to:

> The base address of the min-state save area must minimally be aligned on a 512-byte boundary with, but larger alignments are allowed"

3. Change the PAL_MC_REGISTER_MEM procedure as shown:

PAL_MC_REGISTER_MEM – Register Memory with PAL for Machine Check and Init (27)

Purpose: Registers a platform dependent location with PAL to which it can save minimal processor state in the event of a machine check or initialization event.

Calling Conv: Static Registers Only

Mode: Physical

Buffer: Not dependent

Arguments:

Argument	Description
index	Index of PAL_MC_REGISTER_MEM within the list of PAL procedures.
address	Physical address of the buffer to be registered with PAL.
size	Unsigned integer indicating the size in kilobytes (KB) of the buffer passed. This input argument is only required when passing in a size greater than 4KB. The implementation indicates when a size greater than 4KB is required at the reset hand-off. Refer to Section 11.2.2.1 for more information.
Reserved	0

Returns:

Return Value	Description
status	Return status of the PAL_MC_REGISTER_MEM procedure.
req_size	Returns the required size of the min-state save area in kilobytes (KB) if the 'size' input argument did not match the required size for this implementation.
Reserved	0
Reserved	0

Status:

Status Value	Description
0	Call completed without error
-2	Invalid argument
-3	Call completed with error

Description: PAL places the address passed in the XR0 register, which is used by PAL as the min-state save area

This procedure is used to register with PAL an uncacheable min-state save area memory buffer that is used for machine check and initialization event handling. The size of the min-state save area is either 4KB or a larger size that is indicated in the reset hand-off state described in Section 11.2.2.1. The input argument *size* indicates the size of the min-state save buffer in kilobytes (KB) when it is greater than 4KB. If the *size* input argument does not match the required size, the procedure returns an invalid argument return status and a min-state area is not registered. The procedure will also return the required size of the min-state save area in the *reg_size* return value.

The layout of the min-state save area is defined in Section 11.3.2.3 "Processor Min-state Save Area Layout" on page 2:294. The address passed has a minimum alignment requirement of 512-bytes.

4 Specification Clarifications

1. Clarification of ptc.g release semantics

1. Volume 3, Part I, Section 2.2, “Instruction Descriptions”, ptc.g page: Change the following text in the Description section:

   `ptc.g` has release semantics and is guaranteed to be made visible after all previous data memory accesses are made visible. The memory fence instruction forces all processors to complete the purge prior to any subsequent memory operations. Serialization is still required to observe the side-effects of a translation being removed.

   to:

   `ptc.g` has release semantics and is guaranteed to be made visible after all previous data memory accesses are made visible. Serialization is still required to observe the side-effects of a translation being removed. If it is desired that the `ptc.g` become visible before any subsequent data memory accesses are made visible, a memory fence instruction (`mf`) should be executed immediately following the `ptc.g`.

2. Volume 2, Part I, Section 4.4.7, "Sequentiality Attribute and Ordering”. Change the following text in the fifth paragraph:

   Global TLB purge instructions (`ptc.g` and `ptc.ga`) follow release semantics on the local processor as well as on remote processors, except with respect to global purge instructions being executed by that remote processor.

   to:

   Global TLB purge instructions (`ptc.g` and `ptc.ga`) follow release semantics on the local processor. They are also broadcast to all other processors in the TLB coherence domain; on each such remote processor, a point is chosen in its program-order execution and a local TLB purge operation is inserted at that point; this local TLB purge operation follows release semantics, except with respect to global purge instructions being executed by that remote processor.

2. Clarification of PAL_MC_ERROR_INFO reporting of uncacheable transactions

1. In Volume 2, Part I, Section 11.3.2.1, “Processor State Parameter (GR 18)”, Table 11-7. For the *cc* field, add the following statement at the end of the text in the description box:

   This bit must not be set for non-cacheable transaction errors.

2. In Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”, PAL_MC_ERROR_INFO page:

   a. In the paragraph beginning "**Cache_Check Return Format**:", add the following sentence immediately after the sentence ending: "...caches in the level_index input argument.":

      The cache_check return format must be used to report errors in cacheable transactions. These errors may also be reported using the bus_check return format if the bus structures can detect these errors.

   b. In the paragraph beginning "**Bus_check Return Format**:", add the following sentence immediately after the sentence ending: "...bus structure as specified in the *level_index* input argument.":

The bus_check return format must be used to report errors in uncacheable transactions. These errors must not be reported using the cache_check return format.

3. Clarification of behavior when ptc.g overlaps a translation register

1. Volume 2, Part II, Section 5.2.2.2.3, "ptc.g, ptc.ga", fourth paragraph from the end, change the following:

   The `ptc.g` instruction does not modify the page tables nor any other memory location. It affects both the local and all remote TC entries in the TLB coherence domain. It does not remove translations from either local or remote TR entries, and if a `ptc.g` overlaps a translation contained in a TR on either the local processor or on any remote processor in the coherence domain, the processor containing the overlapping translation will raise a Machine Check Abort.

   to:

   The `ptc.g` instruction does not modify the page tables nor any other memory location. It affects both the local and all remote TC entries in the TLB coherence domain. It does not remove translations from either local or remote TR entries. If a `ptc.g` overlaps a translation contained in a TR on the local processor, the local processor will raise a Machine Check Abort; if the ptc.g overlaps a translation contained in a TR on any remote processor in the coherence domain, no Machine Check Abort is raised.

4. INT3 Clarifications

1. Volume 2, Part I, Section 5.6, “Interruption Priorities”, Table 5-6. Change the following row:

80	IA-32 Breakpoint (INT 3) trap	IA-32 Exception vector (Debug)

   to:

80	IA-32 Breakpoint (INT 3) trap	IA-32 Exception vector (Break)

2. Volume 2, Part I, Section 7.1, “Debugging”. In the “**Break Instruction fault**” bullet, change the following sentence:

   Execution of the IA-32 INT 3 (break) instruction results in a IA_32_Exception(Debug) trap.

   to:

   Execution of the IA-32 INT 3 (break) instruction results in a IA_32_Exception(Break) trap.

5. Test feature instruction clarifications

1. Volume 1, Part I, Section 4.3.2, “Compare Instructions”.

   a. Change the first paragraph from:

      Predicate registers are written by the following instructions: general register compare (`cmp`, `cmp4`), floating-point register compare (`fcmp`), test bit and test NaT (`tbit`, `tnat`), floating-point class (`fclass`), and floating-point reciprocal approximation and reciprocal square root approximation (`frcpa`, `fprcpa`, `frsqrta`, `fprsqrta`). Most of these compare instructions (all but `frcpa`, `fprcpa`, `frsqrta` and `fprsqrta`) set two predicate registers based on the outcome of the comparison. The setting of the two target registers is described below in Compare Types on page 1:53. Compare instructions are summarized in Table 4-8.

      to:

      Predicate registers are written by the following instructions: general register compare (`cmp`, `cmp4`), floating-point register compare (`fcmp`), test bit and test NaT (`tbit`, `tnat`), test feature (`tf`), floating-point class (`fclass`), and floating-point reciprocal approximation and reciprocal square root approximation (`frcpa`, `fprcpa`, `frsqrta`, `fprsqrta`). Most of these compare instructions (all but `frcpa`, `fprcpa`, `frsqrta` and `fprsqrta`) set two predicate registers based on the outcome of the comparison. The setting of the two target registers is described below in Compare Types on page 1:53. Compare instructions are summarized in Table 4-8.

b. Volume 1, Part I, Section 4.3.2, “Compare Instructions”, Table 4-8, add a new row, just under the row for `tnat`, with the following information:

Mnemonic	Operation
`tf`	Test feature

c. Volume 1, Part I, Section 4.3.2, “Compare Instructions”, at the end of the following paragraph:

The test bit (`tbit`) instruction sets two predicate registers according to the state of a single bit in a general register (the position of the bit is specified by an immediate). The test NaT (`tnat`) instruction sets two predicate registers according to the state of the NaT bit corresponding to a general register.

add the following sentence:

The test feature (`tf`) instruction sets two predicate registers according to whether or not the selected feature is implemented in the processor.

d. Volume 1, Part I, Section 4.3.3, “Compare Types”, Table 4-11, change the row for

`tbit`, `tnat`

to read:

`tbit`, `tnat`, `tf`

2. Volume 1, Part I, Section 3.4.2, "WAW Dependency Special Cases", change the first sentence of the second paragraph from:

The set of compare-type instructions includes: `cmp`, `cmp4`, `tbit`, `tnat`, `fcmp`, `frsqrta`, `frcpa`, and `fclass`.

to:

The set of compare-type instructions includes: `cmp`, `cmp4`, `tbit`, `tnat`, `tf`, `fcmp`, `frsqrta`, `frcpa`, and `fclass`.

6. Clarification of performance counter behavior under halt states

Volume 2, Part I, Section 7.2.3. "Performance Monitor Events". Change the following text:

1. The number of retired instructions. These are defined as all instructions which execute without a fault, including nops and those which were predicated off.

2. The number of processor clock cycles the CPU is in either the NORMAL or LOW-POWER state (see Figure 11-19 on page 2:303).

to:

1. The number of retired instructions. These are defined as all instructions which execute without a fault, including nops and those which were predicated off.

Generic counters configured for this event count only when the processor is in the NORMAL or LOW-POWER state (see Figure 11-19 on page 2:303).

2. The number of processor clock cycles. Generic counters configured for this event count only when the processor is in the NORMAL or LOW-POWER state (see Figure 11-19 on page 2:303).

7. PMI Clarifications

1. Clarifications to Volume 2, Part I, Chapter 5, “Interruptions”.

   a. Volume 2, Part I, Section 5.1, "Interruption Definitions", change the following bullet:

      - **Platform Management Interrupts (PMI)**
        A platform management request to perform functions such as platform error handling, memory scrubbing, or power management has been received by a processor. The PALE_PMI entry point is entered to service the request. Program execution may be resumed at the point of interruption. PMIs are distinguished by unique vector numbers. Vectors 0 through 3 are available for platform firmware use and are present on every processor model. Vectors 4 and above are reserved for processor firmware use. The size of the vector space is model specific.

   to:

      - **Platform Management Interrupts (PMI)**
        A platform management request to perform functions such as platform error handling, memory scrubbing, or power management has been received by a processor. The PALE_PMI entry point is entered to service the request. Program execution may be resumed at the point of interruption. PMIs are distinguished by unique vector numbers. Vectors 0 through 3 are available for platform firmware use and are present on every processor model. Vectors 4 through 15 are reserved for processor firmware use. See section 11.5, "Platform Management Interrupt (PMI)" for details.

   b. Volume 2, Part I, Section 5.8.1, "Interrupt Vectors and Priorities", change the following paragraph:

      PMIs have a separate vector space from external interrupts. PMI vectors 0-3 can be used by platform firmware. PMI vectors 4 and above are reserved for use by processor firmware. Assertion of the processor's PMI pin, when present, results in PMI vector number 0. PMI vector priorities are described in Chapter 11, "Processor Abstraction Layer."

   to:

      PMIs have a separate vector space from external interrupts. PMI vectors 0-3 can be used by platform firmware. PMI vectors 4 through 15 are reserved for use by processor firmware. Assertion of the processor's PMI pin, when present, results in PMI vector number 0. PMI vector priorities are described in "Platform Management Interrupt (PMI)".

   c. Volume 2, Part I, Section 5.8.4.1, “Inter-processor Interrupt Messages”, Table 5-17. At the end of the description for the PMI delivery mode:

      PMI - pend a PMI interrupt for the specified vector to the processor listed in the destination. Allowed PMI vector values are 0-3. All other PMI vector values are reserved for use by processor firmware.

   Add the following:

      See Section 11.5, "Platform Management Interrupt (PMI)" for details.

2. Volume 2, Part I, Section 11.5.1, "PMI Overview":
    a. In the fifth paragraph (just above table 11-13), delete the last sentence:

        "Vectors described as Intel reserved will be ignored by the processor."

    b. In Table 11-13, change the text of second column from the left from:

        Intel Reserved PAL

        to:

        PAL Reserved

    c. In Table 11-13, in the Description column, change both instances of

        Intel Reserved

        to:

        PAL Reserved

3. Volume 2, Part I, Section 11.5.1 "PMI Overview".
    a. Change the first sentence of the first paragraph from:

        PMI is an asynchronous highest-priority external interrupt that encapsulates...

        to:

        PMI is an asynchronous interrupt that encapsulates...

    b. Change the second sentence of the third paragraph from:

        PMI events are the highest priority external interrupts...

        to:

        PMI events are asynchronous interrupts higher priority than all external interrupts...

8. PAL_MC_ERROR_INJECT Clarifications

1. Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”. In the PAL_MC_ERROR_INJECT procedure, change the 'err_data_buffer' argument description from:

    64-bit physical address of a buffer providing additional parameters for the requested error. The address of this buffer must be 8-byte aligned.

    to:

    Unsigned 64-bit integer specifying the address of the buffer providing additional parameters for the requested error. The address of this buffer must be 8-byte aligned.

2. Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications”. In the PAL_MC_ERROR_INJECT specification, Table 11-95 "err_struct_info - Register File."
    a. For the “reg_num” field, change the “Bits” value from:

        11:5

        to:

        12:5

    b. For the “Reserved” field just under “reg_num” change the "Bits" value from:

        31:12

        to:

        31:13

9. Min-state Save Area Clarifications

1. In Volume 2, Part I, Section 11.3.2.3, “Processor Min-state Save Area Layout”. Add the following text, figure and table after Figure 11.14:

   The NaT bits stored in the first entry of the min-state save area have the following layout.

Figure 11-15. Min-state Save Area NaT Bits

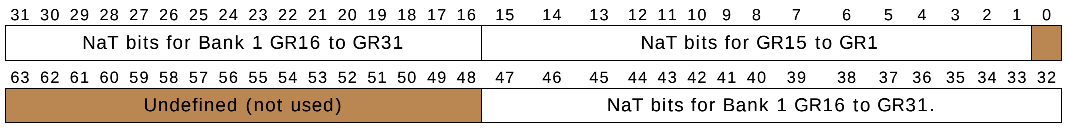


Table 11-9. Min-state Save Area Nat Bits Description

Bits	Description
0	Undefined (not used)
15:1	NaT bits for GR15 to GR1. Bit 1 represents GR1 and subsequent bits follow the ascending pattern
31:16	NaT bits for Bank 0 GR16 to GR31. Bit 16 represents Bank 0 GR16 and subsequent bits follow the ascending pattern.
47:32	NaT bits for Bank 1 GR16 to GR31. Bit 32 represents Bank 1 GR16 and subsequent bits follow the ascending pattern.
63:48	Undefined (not used).

2. Volume 2, Part I, Section 11.3.2, “PALE_CHECK Exit State”. Change the first sentence from:

   The state of the processor on exiting PALE_CHECK is:

   to:

   The state of the processor on exiting PALE_CHECK is listed below. For registers described as being saved to the min-state save area and available for use, the actual values in these registers are undefined unless specifically stated otherwise.

3. Volume 2, Part I, Section 11.4.2, “PALE_INIT Exit State”. Change the first sentence from:

   The state of the processor on exiting PALE_INIT is:

   to:

   The state of the processor on exiting PALE_INIT is listed below. For registers described as being saved to the min-state save area and available for use, the actual values in these registers are undefined unless specifically stated otherwise.

10. Semaphore Code Corrections

1. Volume 2, Part II, Section 2.4.1 "Spin Lock". In Figure 2-4 change the instruction:

   ```
   cmpxchg8.acq  r1 = [lock], r2 ;; // attempt to grab lock
   ```

   to:

   ```
   cmpxchg8.acq  r1 = [lock], r2, ar.ccv ;; // attempt to grab lock
   ```

2. Volume 2, Part II, Section 2.4.3, "Dekker's Algorithm". In Figure 2-6, change the "cmp.eq" in the following code sequence from:

   ```
            ld8 r2 = [flag_you] ;;// is other's flag 0?
            cmp.eq  p1, p0 = 0, r2
   (p1)     br.cond.spnt cs_skip ;;// if not, resource in use
   ```

to “cmp.ne” as shown:

```
          ld8 r2 = [flag_you] ;;// is other's flag 0?
          cmp.eq  p1, p0 = 0, r2
   (p1)   br.cond.spnt cs_skip ;;// if not, resource in use
```

5 Documentation Changes

1. Revision 2.2 Documentation Changes

1. Volume 2, Part I, Section 8.3, “Interruption Vector Definition, Table 8-2. Change the second to last vector from:

   External Interrupt vector

   to:

   Virtual External Interrupt vector

2. Volume 2, Part I, Section 8.3, “Interruption Vector Definition, Page 2:162, Table 8-4. Add the following vector to the table:

Vector Name	Offset	Page
Virtual External Interrupt vector	0x3400	2:177

3. Volume 2, Part I, Section 11.10.3, “PAL Procedure Specifications, PAL_PSTATE_INFO page, Figure 11-67. Change the name of the field {10:5} from “ddit” to “ddid”.

§