Intel® Itanium™ Architecture Software Developer's Manual

Specification Update

August 2001

Notice: The Intel® Itanium processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this specification update.

Document Number: 248699-005

Contents

This page intentionally left blank.

Revision History

Revision Number	Description	Date
1.0	Initial version of this document.	April 2000
2.0	Added changes to performance monitoring Section (Section 7.8, Volume 4 Added a clarification to class pr-writers-int in Table A-5 of Volume 2 Added a clarification to Section 4.4.6.1 of Volume 2	October 2000
3.0	Added a VHPT walk and forward progress change (Section 4.1.1.2, Volume 2) Revised Chapter 7 of Volume 4, Performance Monitoring Events (new Section 7.6.5, Frontside Bus; added bus monitors to Section 7.8, Event List; misc. changes and fixes) Added a faults in ld.c that hits ALAT clarification (Section 4.4.5.3.1, Volume 1) Added an IA-32 IBR/DBR match clarification (Section 7.1.1, Volume 2) Added an IA-32 CPUID clarification (p. 5-71 of Volume 3) Added ISR figure changes on pp. 8-5, 8-26, 8-33 and 8-36 of Volume 2)	December 2000
4.0	Added a change to PAL_CACHE_FLUSH return argument (Section 11.8.3, Volume 2) Added a change to PAL self-test Control and PAL_A procedure requirement change (Section 11.2, Volume 2) Added clarifications to PAL_CACHE_FLUSH (Section 11, Volume 2) Added a clarification to non-speculative reference (Section 4.4.6, Volume 2) Added clarifications to RID and Preferred Page Size usage (Section 4.1, Volume 2) Added clarifications to VHPT read atomicity (Section 4.1, Volume 2) Added clarifications to IIP and WC flush (Section 4.4.5, Volume 2) Revised IBR and DBR addressing (Section 6.2.4, Volume 4) Revised figures for extract, deposit, and alloc instructions (Section 2.2, Volume 3) Revised RSE and PMC typographical errors (Section 6.4, Volume 2) Revised DV table (Section A.4, Volume 2)	March 2001
5.0	Added a change regarding memory attribute transitions (Section 4.4, Volume 2) Added a change regarding MCA for WC/UC aliasing (Section 4.4.1, Volume 2) Added a change regarding bus lock deprecation (Section 3.3.4.1, Section 10.6.8, Section 11.8.3; Volume 2) Added changes regarding PAL_PROC_GET/SET_FEATURES (Section 11.8.3, Volume 2) Added changes regarding Split PAL_A architecture (Section 11.1.6, Volume 2) Added a clarification regarding simple barrier synchronization (Section 13.4.2, Volume 2) Added a clarification regarding limited speculation (Section 4.4.6, Volume 2) Added a clarification regarding RCPPS, RCPSS, RSQRTPS, and RSQRTSS (Section 7.12, Volume 3) Added a clarification regarding PAL memory accesses and restrictions (Section 11.9, Volume 2) Added a clarification regarding PSP validity on INITs from PAL_MC_ERROR_INFO (Section 11.8.3, Volume 2) Added a clarification regarding speculation attributes (Section 4.4.6, Volume 2) Added PAL_A FIT entry, PAL_VM_TR_READ, PSP, PAL_VERSION clarifications (Sections 11.8.3 and 11.3.2.1, Volume 2) Added TLB searching clarifications (Section 4.1, Volume 2) Added IA-32 related document changes (Volume 1: Section 6.2.5.4, Section 6.2.3, Section 6.2.4, Section 6.2.5.3; Volume 2: Section 10.3, Section 10.3.2, Section 10.3.2, Section 10.3.3.1, Section 10.10.1; Volume 3: Section 5.3; Volume 4: Section 8) Added load instructions change (Section 4.4.1, Volume 1)	August 2001

Revision Number	Description	Date
5.0	Added tak, tpa change Section 2.2, Volume 3) Added IPSR.ri and ISR.ei changes (Volume 2) Added miscellaneous performance monitoring events changes (Volume 4, Chapter 7)	August 2001

Preface

This document is an update to the specifications contained in the Affected Documents/Related Documents in the table below. This document is a compilation of specification changes, specification clarifications and document changes. It does not cover errata.

Affected Documents/Related Documents

Title	Document #
Intel® Itanium™ Architecture Software Developer's Manual, Volume 1: Application Architecture	245317
Intel® Itanium™ Architecture Software Developer's Manual, Volume 2: System Architecture	245318
Intel® Itanium™ Architecture Software Developer's Manual, Volume 3: Instruction Set Reference	245319
Intel® Itanium™ Architecture Software Developer's Manual, Volume 4: Itanium™ Processor Programmer's Guide	245320

Nomenclature

Specification Changes are modifications to the current published specifications for the Itanium™ processor. These changes will be incorporated in the next release of the specifications.

Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. 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*.

Summary Table of Changes

The following tables indicate the specification changes, specification clarifications, or documentation changes which apply to the *Intel® Itanium™ Architecture Software Developer's Manual*.

Specification Changes

No.	Page	SPECIFICATION CHANGES
1	10	Volume 2: Memory attribute transition change
2	14	Volume 2: MCA for WC/UC aliasing change
3	14	Volume 2: Bus lock deprecation change
4	15	Volume 2: PAL_PROC_GET/SET_FEATURES change
5	16	Volume 2: Split PAL_A architecture changes
6	20	Volume 2: PAL_CACHE_FLUSH return argument addition
7	20	Volume 2: PAL self-test Control and PAL_A procedure requirement change
8	26	Volume 2: VHPT walks and forward progress
9	27	Revised Chapter 7 of Volume 4, Performance Monitoring Events (text included at end of this update)

Specification Clarifications

No.	Page	SPECIFICATION CLARIFICATIONS
1	28	Volume 2: Barrier clarification
2	28	Volume 2: Limited speculation clarifications
3	29	Volume 3 and Volume 4: RCPPS/RCPSS and RSQRTPS/RSQRTSS clarifications
4	30	Volume 2: PAL code memory accesses and restrictions clarification
5	31	Volume 2: PSP validity on INITs from PAL_MC_ERROR_INFO clarification
6	31	Volume 2: Non-speculative memory, misalignment and multiple accesses clarification
7	31	Volume 2: PAL_A FIT entry, PAL_VM_TR_READ, PSP, and PAL_VERSION clarifications
8	32	Volume 2: TLB searching clarification
9	34	Volume 2: PAL_CACHE_FLUSH clarifications
10	36	Volume 2: Non-speculative reference clarification
11	36	Volume 2: RID and Preferred Page Size usage clarification
12	36	Volume 2: VHPT read atomicity clarification
13	38	Volume 2: IIP and WC flush clarifications
14	38	Volume 1: Faults in ld.c that hits ALAT clarification
15	38	Volume 2: IA-32 IBR/DBR match clarification
16	39	Volume 3: IA-32 CPUID clarification
17	39	Volume 2: Table A-5 class pr-writers-int
18	39	Volume 2: Section 4.4.6.1 PAL_MC_DRAIN procedure only causes cache line writeback transactions to be forced onto the bus, and does not guarantee that they reached main memory

Documentation Changes

No.	Page	DOCUMENTATION CHANGES
1	40	Volume 1, Volume 2, Volume 3, and Volume 4: IA-32 related documentation changes
2	42	Volume 1: Load instructions change
3	42	Volume 3: tak, tpa change
4	43	Volume 2: IPSR.ri and ISR.ei changes
5	44	Volume 4: Miscellaneous Chapter 7 “Performance Monitoring Events” typographical errors (text included at end of this update)
6	44	Volume 4: IBR and DBR addressing typographical errors
7	45	Volume 3: Figure changes for extract, deposit, and alloc instructions
8	46	Volume 2: RSE and PMC typographical errors
9	47	Volume 2: DV table typographical error
10	47	Volume 2: ISR figure and wording changes
11	47	Volume 2: ISR figure change on the Lower-privilege Transfer Trap vector page (new p. 8-36 of Volume 2)
12	48	Volume 4: Figure 6-17 and Figure 6-19 bit typo
13	48	Volume 4: Chapter 6 the eight stall and flush reasons

Specification Changes

1. Volume 2: Memory attribute transition change

1) On p. 4-38, create a new Section 4.4.11, titled "Memory Attribute Transition" (at the end of Section 4.4, before Section 4.5) and add this paragraph just after the new section heading:

> If software modifies the memory attributes for a page, it must perform explicit actions to ensure that subsequent reads and writes using the new attribute will be coherent with prior reads and writes that were performed with the old attribute. Processors may have separate buffers for coalescing, uncacheable and cacheable references, and these buffers need not be coherent with each other.

2) On p. 4-27, in Section 4.4.1, change the second paragraph on the page from:

> If software modifies the memory attributes for a page, software must flush any processor cache copies with the FlushCache (`fc`) instruction for the following memory attribute changes: speculative/non-speculative, cacheable/uncacheable (for transitions from cacheable to uncacheable), and coherency. Software must flush any coalescing buffers if a page is changed from coalescing to any other attribute. See "Coalescing Attribute" on page 4-29.
>
> to:
>
> If software modifies the memory attributes for a page, it must follow the attribute transition requirements in Section 4.4.11.

3) After the new Section 4.4.11, create a new Section 4.4.11.1, titled "Virtual Addressing Memory Attribute Transition" and add this paragraph:

> To change a virtually-addressed page from one attribute to another, software must perform the following sequence. (The address of the page whose attribute is being modified is referred to as "X")
>
> Note that this sequence is ONLY required if the new mapping and the old mapping do not have the same memory attribute.
>
> On the processor initiating the transition, perform the following steps 1-3:
>
> 1. PTE[X].p = 0 // Mark page as not present
>
>   This prevents any processors from reading the old mapping (with the old attribute) from the VHPT after this point.
>
> 2. ptc.ga [X] ;; // Global shootdown and ALAT invalidate for the
>   // entire page
>
>   This removes the mapping from all processor TC's in the coherence domain, and it forces all processors to flush any pending WC or UC stores from write buffers.
>
> 3. mf ;; // Ensure visibility of ptc.ga to local data stream
>   srlz.i ;; // Ensure visibility of ptc.ga to local instruction
>   // stream
>
>   After step 3, no processor in the coherence domain will initiate new memory references or prefetches to the old translation. Note, however, that memory references or prefetches initiated to the old translation prior to step 2 may still be in progress after step 3. These outstanding memory references and prefetches may return instructions or data which may be placed in the processor cache hierarchy; this behavior is implementation-specific.

If the new memory attribute is an uncacheable attribute, and if the old attribute was cacheable (or if it is not known at this point in the code sequence what the old attribute was), then software must drain any current prefetches and ensure that any cached data from the page is removed from caches. To do this, software must perform steps 4-10. If the new memory attribute is cacheable, then software may skip steps 4-10, and go straight to step 11.

4. Call PAL_PREFETCH_VISIBILITY

   Call PAL_PREFETCH_VISIBILITY with the the input argument 'trans_type' equal to zero to indicate that the transition is for virtual memory attributes. The return argument from this procedure informs the caller if this procedure call is needed on remote processors or not. If this procedure call is not needed on remote processors, then software may skip the IPI in step 5 and go straight to step 6 below.

5. Using the IPI mechanism defined in "Inter-Processor Interrupt Messages" on page 5-32 to reach all processors in the coherence domain, perform step 4 above on all processors in the coherence domain, and wait for all PAL_PREFETCH_VISIBILITY calls to complete on all processors in the coherence domain before continuing.

   After steps 4 and 5, no more new instruction or data prefetches will be made to page "X" by any processor in the coherence domain. However, processor caches in the coherence domain may still contain "stale" data or instructions from prior prefetch or memory references to page "X".

6. Insert a temporary UC translation for page "X"

7. ```
   fc [X]              // flush all processor caches in the coherence
                       // domain
   fc [X+32]
   fc [X+64]
   ...                 // ... for all of page "X" (page size = ps)
   fc [X+ps-32] ;;
   sync.i ;;           // Ensure cache flushes are also seen by
                       // processors' instruction fetch
   ```

   After step 7, all flush cache instructions initiated in step 7 are visible to all processors in the coherence domain, i.e., no processor in the coherence domain will respond with a cache hit on a memory reference to an address belonging to page "X".

8. Purge the temporary UC translation from the TLB

9. Call PAL_MC_DRAIN

10. Using the IPI mechanism defined in "Inter-Processor Interrupt Messages" on page 5-32 to reach all processors in the coherence domain, perform step 9 above on all processors in the coherence domain, and wait for all PAL_MC_DRAIN calls to complete on all processors in the coherence domain before continuing.

    This further guarantees that any cache lines containing addresses belonging to page [X] have been evicted from all caches in the coherence domain and * forced onto the bus *. Note that this operation does not ensure that the cache lines have been written back to memory.

11. Insert the new mapping with the new memory attribute

4) Add a new Section 4.4.11.2 titled "Physical Addressing Attribute Transition - Disabling Prefetch/Speculation and Removing Cacheability" with 3 new headings, just after the new Section 4.4.11.1 "Virtual Addressing Memory Attribute Transition" and add this body to the newly-created section:
   ```

When a non-speculative reference is made to a physical address with the WBL attribute, the 4K page containing that address becomes speculatively accessible. This allows the processor that made the non-speculative reference to subsequently make speculative references to this page. (See the description of limited speculation in Section 4.4.6.)

If the same physical memory is then to be accessed with the UC attribute, software must first make all such addresses speculatively inaccessible and flush any cached copies from the cache. Otherwise, an uncacheable reference may hit in cache, causing a Machine Check abort.

Also, if physical memory is to be removed from the system, or if physical memory is to be re-configured in such a way that some physical address X, which used to correspond to some portion of memory will now corresponds to nothing in the system, software take these same actions. Otherwise, the processor may initiate a speculative prefetch after the memory has been removed or re-configured, causing a Machine Check abort.

On the processor initiating the transition, perform the following steps:

1. Call PAL_PREFETCH_VISIBILITY

   Call PAL_PREFETCH_VISIBILITY with the the input argument 'trans_type' equal to one to indicate that the transition is for physical memory attributes. This PAL call terminates the processor's rights to make speculative references to any limited speculation pages (i.e., it makes all WBL pages speculatively inaccessible - see the discussion on limited speculation in Section 4.4.6.)

   The return argument from this procedure informs the caller if this procedure call is needed on remote processors or not. If this procedure call is not needed on remote processors, then software may skip the IPI in step 2 and go straight to step 3 below.

2. Using the IPI mechanism defined in "Inter-Processor Interrupt Messages" on page 5-32 to reach all processors in the coherence domain, perform step 1 above on all processors in the coherence domain, and wait for all PAL_PREFETCH_VISIBILITY calls to complete on all processors in the coherence domain before continuing.

   On the processor initiating the disabling process, continue the sequence:

3.
```
fc [X]              // flush all processor caches in the coherence
                    // domain
fc [X+32]
fc [X+64]
...                 // ... for all of page "X" (page size = ps)
fc [X+ps-32] ;;
sync.i ;;           // Ensure cache flushes are also seen by
                    // processors' instruction fetch
```

   After step 3, all flush cache instructions initiated in step 3 are visible to all processors in the coherence domain, i.e., no processor in the coherence domain will respond with a cache line hit on a memory reference to an address belonging to page "X"."

4. Call PAL_MC_DRAIN

5. Using the IPI mechanism defined in "Inter-Processor Interrupt Messages" on page 5-32 to reach all processors in the coherence domain, perform step 4 above on all processors in the coherence domain, and wait for all PAL_MC_DRAIN calls to complete on all processors in the coherence domain before continuing.

   This further guarantees that any cache lines containing addresses belonging to page [X] have been evicted from all caches in the coherence domain and * forced onto the bus *. Note that this operation does not ensure that the cache lines have been written back to memory.

   This sequence ensures that speculation and prefetch are disabled for all WBL pages, that all outstanding prefetches have completed, and that the caches have been flushed. It may

also be necessary to take additional platform-dependent steps to ensure that all cache write-back transactions have completed to memory before removing or re-configuring physical memory.

5) Delete current WB->UC transition section. Delete Section 4.4.6.1 "Disabling Prefetch and Removing Cacheability".

6) On p. 11-95, PAL_PREFETCH_VISIBILITY:

6a) Change the purpose section from:

Used in the architected sequence in Section 4.4.6.1 "Disabling Prefetch and Removing Cacheability" to transition a page (or set of pages) from a cacheable, speculative attribute to an uncacheable attribute.

to:

Used in the architected sequences for memory attribute transitions described in Section 4.4.11 "Memory Attribute Transition" to transition a page (or set of pages) from a one memory attribute to another.

6b) Add a new argument “trans_type” with description “Unsigned integer specifying the type of memory attribute transition that is being performed.”

6c) Change the description section from:

This call is intended to be used only in the architected sequence in Section 4.4.6.1 "Disabling Prefetch and Removing Cacheability". Use of this procedure outside the context of this sequence results in undefined behavior.

After a successful return from this procedure in the aforementioned architected sequence, all prefetches that were initiated by the processor to the cacheable, speculative translation prior to the call will either not be cached; have been aborted; or are visible to subsequent fc instructions. (from both the local processor and from remote processors)

If the processor implementation does not require this call on remote processors in this sequence, this procedure will return a 1 upon successful completion.

A return value of 0 upon successful completion of this procedure is an indication to software that the processor implementation requires that this call be performed on all processors in the coherence domain to make prefetches visible in this sequence.

These return code can be used to tune the architected sequence to the particular system on which is running; see Section 4.4.6.1 "Disabling Prefetch and Removing Cacheability" for details.

to:

This call is intended to be used only in the architected sequences described in Section 4.4.11 "Memory Attribute Transition". Use of this procedure outside the context of this sequence results in undefined behavior.

The 'trans_type' input indicates if a user is transitioning virtual addressing memory attributes (input value of 0) or physical addressing memory attributes (input value of 1). All other values are reserved.

This procedure, when used for transitioning virtual memory attributes, will ensure that all prefetches that were initiated by the processor to the cacheable, speculative memory prior to the call, will either not be cached; have been aborted; or are visible to subsequent fc instructions. (from both the local processor and from remote processors)

This procedure when used for transitioning physical memory attributes will ensure that all prefetches that were initiated by the processor to the cacheable, limited speculative memory prior to the call, will either not be cached; have been aborted; or are visible to subsequent fc instructions. (from both the local processor and from remote processors) It will also terminate the ability for the processor to make speculative references to any

limited speculation pages. For the processor to make any speculative reference to a limited speculation page after this call, there must be a non-speculative reference made to that page after this call.

If the processor implementation does not require this procedure call to be made on remote processors in the sequences, this procedure will return a 1 upon successful completion.

A return value of 0 upon successful completion of this procedure is an indication to software that the processor implementation requires that this call be performed on all processors in the coherence domain to make prefetches visible in the sequences.

These return code can be used to tune the architected sequence to the particular system on which is running; see Section 4.4.11 "Memory Attribute Transition" for details.

2. Volume 2: MCA for WC/UC aliasing change

On p. 4-27, Section 4.4.1, fourth paragraph, remove the first bullet.

3. Volume 2: Bus lock deprecation change

1) On p. 3-13, Section 3.3.4.1, in Table 3-5, change the 'lc' field description to:

IA-32 Lock Check enable - When 1, and an IA-32 atomic memory reference is defined as requiring a read-modify-write operation external to the processor under an external bus lock, an IA-32_Intercept(Lock) is raised. (IA-32 atomic memory references are defined to require an external bus lock for atomicity when the memory transaction is made to non-write back memory or are unaligned across an implementation specific non-supported alignment boundary.) When 0, and an IA-32 atomic memory reference is defined as requiring a read-modify-write operation external to the processor under external bus lock, the processor may either execute the transaction as a series of non-atomic transactions or perform the transaction with an external bus lock, depending on the processor implementation. IA-64 semaphore accesses ignore this bit. All unaligned IA-64 semaphore references generate an Unaligned Data Reference fault. All IA-64 semaphore references made to memory that is neither write-back cacheable nor a NaTPage result in an Unsupported Data Reference fault.

2) On p. 10-22 in Section 10.6.8 “Atomic Operations”, replace the last three paragraphs from:

If an IA-32 locked atomic operation requires the processor to initiate a read-modify-write operation external to the processor under external bus lock and if DCR.lc is set to 1,

...

For IA-32 semaphores, atomicity to uncached memory areas (UC) is platform specific, atomicity can only be ensured by the platform design and can not be enforced by the processor.

to:

If an IA-32 locked atomic operation is defined as requiring the a read-modify-write operation external to the processor under external bus lock and if DCR.lc is set to 1, an IA-32_Intercept(Lock) fault is generated. (IA-32 atomic memory references are defined to require an external bus lock for atomicity when the memory transaction is made to non-write back memory or are unaligned across an implementation specific non-supported alignment boundary.) If DCR.lc is set to 0, the processor may either execute the transaction as a series of non-atomic transactions or perform the transaction with an external bus lock, depending on the processor implementation. For processor implementations that do support external bus locks, software must ensure that the Bus Lock Mask bit is set to one, in order to ensure atomicity of these IA-32 operations when DCR.lc =0. The Bus Lock Mask bit is a feature controllable by the PAL_BUS_SET_FEATURES procedure. (See Table 11-24 for more information)

If the processor supports external bus locks, unaligned IA-32 atomic references are supported, but their usage is strongly discouraged since they are typically performed outside the processor's cache which can severely degrade performance of the system. IA-32 external bus locks are not supported on all processor implementations.

For IA-32 semaphores, atomicity to uncached memory areas (UC) is platform specific, atomicity can only be ensured by the platform design and can not be enforced by the processor.

3) On p. 11-40, Section 11.8.3, in Table 11-24, change the class of bus lock mask bit, bit 30, from 'Req.' to 'Opt.'.

4. Volume 2: PAL_PROC_GET/SET_FEATURES change

1) On p. 11-96, Section 11.8.3,

a) change the Arguments section of PAL_PROC_GET_FEATURES to:

Argument	Description
index	Index of PAL_PROC_GET_FEATURES within the list of PAL procedures.
Reserved	0
feature_set	Feature set information is being requested for.
Reserved	0

b) change the Status section of PAL_PROC_GET_FEATURES to:

Status Value	Description
1	Call completed without error; The *feature_set* passed is not supported but a *feature_set* of a larger value is supported
0	Call completed without error
-2	Invalid argument
-3	Call completed with error
-8	*feature_set* passed is beyond the maximum *feature_set* supported

c) change the Description section of PAL_PROC_GET_FEATURES to:

PAL_PROC_GET_FEATURES and PAL_PROC_SET_FEATURES procedure calls are used together to describe current settings of processor features and to allow modification of some of these processor features.

The *feature_set* input argument for PAL_PROC_GET_FEATURES describes which processor *feature_set* information is being requested. Table 11-53 describes processor *feature_set* zero. The *feature_set* values are split into two categories: architected and implementation-specific. The architected *feature_sets* have values from 0-15. The implementation-specific *feature_sets* are values 16 and above. The architected *feature_sets* are described in this document. The implementation-specific *feature_sets* are described in processor-specific documentation.

This procedure will return an invalid argument if an unsupported architectural *feature_set* is passed as an input. Implementation-specific *feature_sets* will start at 16 and will expand in an ascending order as new implementation-specific *feature_sets* are added. The return *status* is used by the caller to know which implementation-specific *feature_sets* are currently supported on a particular processor.

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.

In Table 11-53, the *class* field indicates whether a feature is required to be available (*Req.*) or is optional (*Opt.*). The *control* field indicates which features are required to be controllable. *Req.* indicates that the feature must be controllable, *Opt.* indicates that the feature may optionally be controllable, and *No* indicates that the feature cannot be controllable. The *control* field applies only when the feature is available. The sense of the bits is chosen so that for features which are controllable, the default hand-off value at exit from PALE_RESET should be 0. PALE_CHECK and PALE_INIT will not modify these features.

2) In Table 11-53, add a new bit:

55	Opt.	Req.	Enable external notification when the processor detects hardware errors caused by environmental factors that could cause loss of deterministic behavior of the processor. When 1, this bit will enable external notification, when 0 external notification is not provided. The type of external notification of these errors is processor-dependent. A loss of processor deterministic behavior is considered to have occurred if these environmentally induced errors cause the processor to deviate from its normal execution and eventually causes different behavior which can be observed at the processor bus pins. Processor errors that do not have this effects (i.e., software induced machine checks) may or may not be promoted depending on the processor implementation.

3) On p. 11-98, Section 11.8.3,

a) change the Argument section of PAL_PROC_SET_FEATURES to:

Argument	Description
index	Index of PAL_PROC_SET_FEATURES within the list of PAL procedures.
feature_select	64-bit vector denoting desired state of each feature (1=select, 0=non-select).
feature_set	Feature set to apply changes to. See PAL_PROC_GET_FEATURES for more information on feature sets.
Reserved	0

b) change the Status section of PAL_PROC_SET_FEATURES to:

Status Value	Description
1	Call completed without error; The *feature_set* passed is not supported but a *feature_set* of a larger value is supported
0	Call completed without error
-2	Invalid argument
-3	Call completed with error
-8	*feature_set* passed is beyond the maximum *feature_set* supported

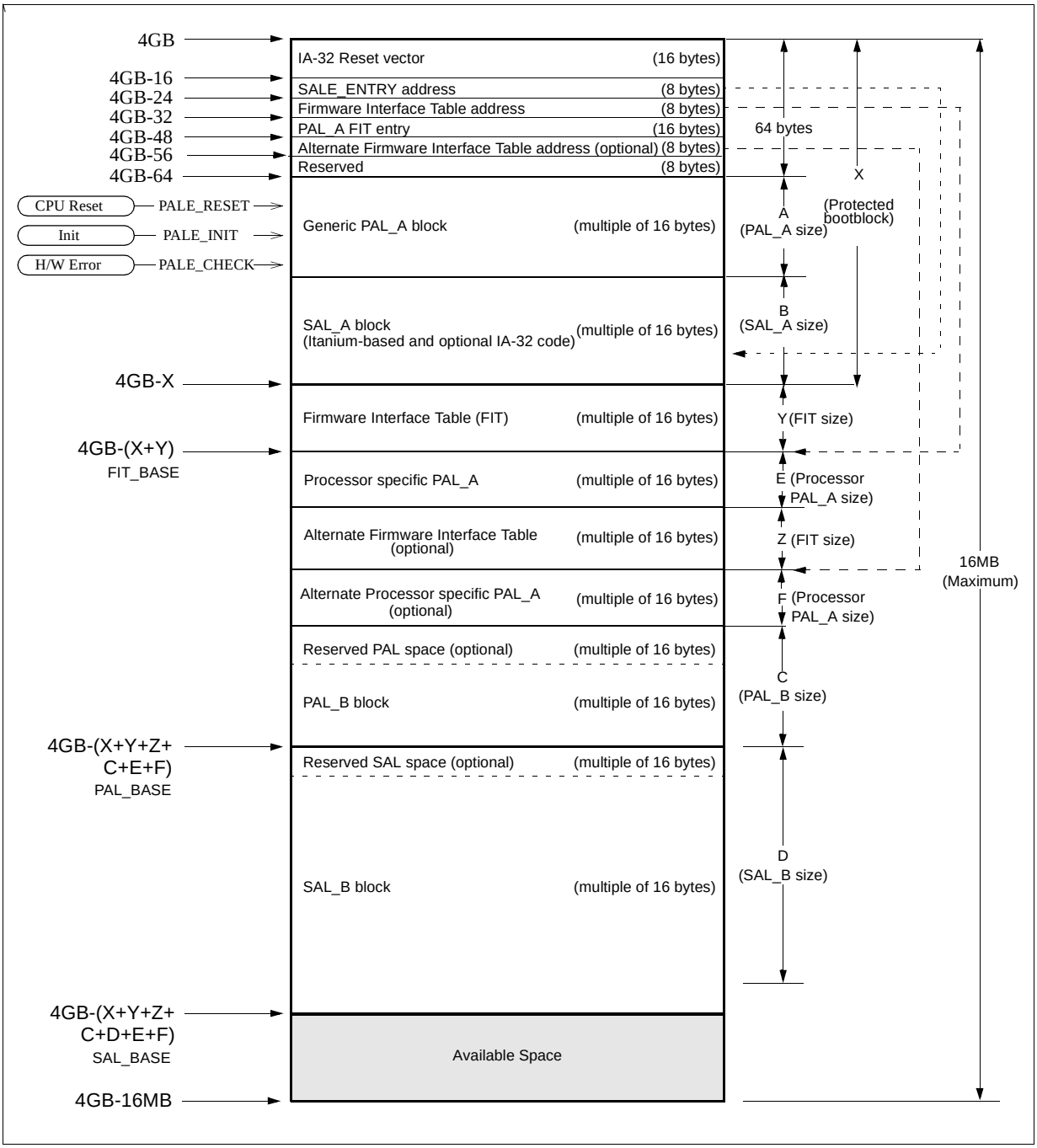
5. Volume 2: Split PAL_A architecture changes

1) On p. 11-5, Section 11.1.6, change the first paragraph to:

The firmware address space occupies the 16 MB region between 4 GB - 16 MB and 4 GB (addresses 0xFF00_0000 through 0xFFFF_FFFF). There are two primary layouts of this address space. The first version is shown in Figure 11-4 and the second version is shown in Figure 11-5. The first version has one PAL_A component. This layout allows for robust recovery of PAL_B and SAL_B components. This layout is useful for cases where PAL_A will not need to be upgraded. The second version splits the PAL_A block into two components. The first component is referred to as the generic PAL_A and the second component is the processor specific PAL_A. Splitting the PAL_A up in this manner allows for a robust upgrade of the processor specific PAL_A firmware as well as the PAL_B and SAL_B components. This is very useful if a platform is designed to support multiple processor generations which would require a PAL_A upgrade when the new processor generation is released. The generic

PAL_A which resides in the Protected Boot Block will work across processor generations for a given platform. The processor specific PAL_A resides outside the Protected Boot Block and works for a specific processor generation.

2) Following Figure 11-4, add a new Figure 11-5 titled “Firmware Address Space with processor specific PAL_A components”:



3) On p. 11-5, Section 11.1.6, change the paragraph and bullets preceeding Figure 11-4 to:

The firmware address space is shared by SAL and PAL. Some of the SAL/PAL boundaries are implementation dependent. The address space contains the following regions and locations.

- The 16 bytes at 0xFFFF_FFF0 (4GB-16) contain IA-32 Reset Code.
- The 8 bytes at 0xFFFF_FFE8 (4GB-24) contain the physical address of the SALE_ENTRY entrypoint.
- The 8 bytes at 0xFFFF_FFE0 (4GB-32) contain the physical address of the Firmware Interface Table.
- The 16 bytes at 0xFFFF_FFD0 (4GB-48) contain the FIT entry for the PAL_A (or generic PAL_A in the split PAL_A model) code provided by the processor vendor. The format of this FIT entry is described in Figure 11-7.
- The 8 bytes at 0xFFFF_FFC8 (4GB-56) contains the physical address of the alternate Firmware Interface Table. This pointer is optional and is only needed if the firmware contains an alternate FIT table. If no alternate FIT table it provided a value of 0x0 should be encoded in this entry.
- The 8 bytes at 0xFFFF_FFC0 (4GB-64) are zero-filled and reserved for future use.
- PAL_A code (also known as generic PAL_A code in split PAL_A model) resides below 0xFFFF_FFC0. This area contains the hardware-triggered entrypoints PALE_RESET, PALE_INIT, and PALE_CHECK. In the model where PAL_A is not split, the PAL_A code will perform any processor-specific initialization needed in order for SAL to perform a firmware recovery. In the split PAL_A model, the generic PAL_A will search the FIT table(s) to find the processor-specific PAL_A code. It will then branch to this code to perform the processor-specific initialization needed in order for SAL to perform a firmware recovery. The PAL_A code area is a multiple of 16 bytes in length.
- SAL_A code occupies the region immediately below the PAL_A code. This area contains the SALE_ENTRY entrypoint as well as optional implementation-independent firmware update code. The SAL_A code area is a multiple of 16 bytes in length.
- The collection of regions above from the beginning of the SAL_A code to 4GB is called the Protected Bootblock. The size of the Protected Bootblock is SAL_A size + PAL_A size + 64.
- The Firmware Interface Table (FIT) comprises of 16-byte entries containing starting address and size information for the firmware components. The FIT is generated at build time, based on the size and location of the firmware components. Optionally, an alternate FIT may be included in the firmware. The alternate FIT will only be used if the primary FIT failed its checksum. In the split PAL_A model, this allows the generic PAL_A firmware to find the processor-specific PAL_A component(s), even if the primary FIT is corrupt. This feature allows hand-off to the SAL recovery code, even if there is a primary FIT checksum failure.
- The processor-specific PAL_A contains the code that is required to be run before handing off to SAL for a firmware recovery check. This component is only available on processors that support a split PAL_A firmware model. One processor specific PAL_A is architecturally required in this model. The firmware may optionally contain two or more processor specific PAL_A components.
- The PAL_B block is comprised of code that is not required to be executed for SAL to perform a firmware recovery update. The PAL_B code area is a multiple of 16 bytes in length. The PAL_B block must be aligned on a 32K byte boundary.

- The remainder of the firmware address space is occupied by SAL_B code. SAL_B may include IA-32 BIOS code. The location of the SAL_B and IA-32 BIOS code within the firmware address space is implementation dependent.

4) On p. 11-7, Section 11.1.6, change the paragraph following Figure 11-5 to:

Each FIT entry contains information for the corresponding firmware component. The first entry contains size and checksum information for the FIT itself and the second entry is used for the PAL_B block. OEMs may use additional entries for other firmware components. FIT entries must be arranged in ascending order by the *type* field, otherwise execution of firmware code will be unpredictable. Multiple FIT entries of the same type are allowed.

5) On p. 11-8, Section 11.1.6, change Table 11-1 “FIT Entry Types” to:

Type	Meaning
0x00	FIT Header
0x01	PAL_B (required)
0x02-0x0D	Reserved
0x0E	Processor Specific PAL_A
0x0F	PAL_A (also generic PAL_A)[a]
0x10-0x7E	OEM-defined
0x7F	Unused Entry

a.Note: The PAL_A FIT entry is is located at 0xFFFF_FFDO (4GB-48) and is not part of the actual FIT table.

---

6. Volume 2: PAL_CACHE_FLUSH return argument addition

On p. 11-42 and 11-45, addition of a new return argument of value 2:

a. Page 11-42, Status section. Add a new return status of value 2:

Status Value	Description
2	Call completed without error, but a PMI was taken during the execution of this procedure.
1	Call has not completed flushing due to a pending external event
0	Call completed without error
-2	Invalid argument
-3	Call completed with error

b. Page 11-45, Status bullet. Add the following new paragraph right under status bullet:

When the call returns a 2, it indicates that the call completed without any errors but that a PMI was taken during the execution of this call. This indicates to the caller that all cache lines that were present in the cache (when the most recent call to PAL_CACHE_FLUSH with a *progress_indicator* of zero) are flushed but that code and data related to handling PMIs may be present in the cache.

7. Volume 2: PAL self-test Control and PAL_A procedure requirement change

On p. 11-9, Section 11.2.1, replace the following:

...PALE_RESET then branches to SALE_ENTRY to determine if a recovery condition exists, which would require an update of the firmware. If it does, SALE_ENTRY performs the update and resets the system. If not, SAL returns to PALE_RESET, which performs a full processor self-test and initialization. PAL may execute IA-32 instructions ...

WITH (new paragraph):

PALE_RESET then branches to SALE_ENTRY to determine if a recovery condition exists, which would require an update of the firmware. If it does, SALE_ENTRY performs the update and resets the system. If no firmware recovery is needed, SAL returns to PALE_RESET to perform the processor self-tests and initialization. SAL can control the length and coverage of the PAL processor self-test by examining and modifying the self-test control word passed to SAL at the firmware recovery hand-off state. Please see Section 11.2.3 for more information on the self-test control word.

The PAL processor self-tests are split into two phases. The first phase is written to test processor features that do not require external memory to be present to execute correctly. These tests are automatically run when SAL returns to PAL after the branch to SALE_ENTRY for a firmware recovery check. This section is referred to as phase one of processor self-test and they are generally run early during the processor boot process. The second phase is written requiring that external memory is available to execute correctly. These tests are run when a call to the PAL procedure PAL_TEST_PROC is made with the correct parameters set up. These tests are referred to as phase two of processor self-test since they are usually run later in the processor boot process after external memory has been initialized on the platform.

PAL may execute IA-32 instructions ....

On p. 11-9, Section 11.2.2, PALE_RESET Exit State. Modify GR34 bullet as follows:

FROM:

... needed for SALE_ENTRY to perform firmware recovery will be available. These procedures are PAL_PLATFORM_ADDR, PAL_PROC_GET_FEATURES (to view current settings), PAL_PROC_SET_FEATURES (enable/disable cache), PAL_CACHE_INIT(level=all, sides-both,restrict-no) and an implementation specific PAL procedure for PAL authentication.

TO:

... needed for SALE_ENTRY to perform firmware recovery will be available. These procedures are PAL_PLATFORM_ADDR and an implementation specific PAL procedure for PAL authentication.

On p. 11-9, Section 11.2.2, PALE_RESET Exit State. Add a GR37 bullet:

- GR37 contains the self-test control word as defined in Section 11.2.3. This control word is processor implementation specific and informs SAL if self-test control is implemented and the number of controllable bits. If self-test control is implemented, PAL will read this value when SAL returns to PAL after firmware recovery check. If the self-test control is not supported, this register will be ignored when SAL returns to PAL after firmware recovery check.

On p. 11-10, Cache and TLB bullets. Change

FROM:

- Cache: The processor internal caches are enabled and invalidated. The caches themselves and the paths from the caches to the processor core have been tested. The path from external memory to the caches have not been tested.

- TLB: The TRs and TCs are initialized with all entries having been invalidated. The TLB is disabled because PSR.it=PSR.dt=PSR.rt=0 and is not available for use until after the second phase of processor self-test. (SEE PAL_TEST_PROCESSOR).

TO:

- Cache: The processor internal caches are enabled and invalidated. Unless directed otherwise by the self-test control word, phase one of the processor self-test verifies the caches themselves and the paths from the caches to the processor core. The path from external memory to the caches cannot be tested until phase two of the processor self-test.
- TLB: The TRs and TCs are initialized with all entries having been invalidated. The TLB is disabled because PSR.it=PSR.dt=PSR.rt=0. The TLBs cannot be fully tested until phase two of the processor self-test.

On p. 11-11, description of the *state* field which is part of the Self-test State Parameter. Change

FROM:

- *state* - a 2-bit field indicating the state of the processor after self-test.

TO:

- *state* - a 2-bit field indicating the state of the processor after self-test. If SAL directed PAL to skip some self tests by modifying the self-test control word, failures related to these self-tests will not be reflected in this state.

On p. 11-12, change the bullet describing FUNCTIONALLY RESTRICTED

FROM:

- The paths between the processor controlled caches and the register files must work during the tests in PAL_RESET, and the entire path from memory through the caches to the register file must work during the tests in PAL_TEST_PROCESSOR

TO:

- The paths between the processor controlled caches and the register files have been shown to work. The path between the processor caches and memory cannot be validated until phase two of the processor self-test invoked by the PAL_TEST_PROC procedure.

On p. 11-12, add a new Section 11.2.3 titled PAL Self-test Control Word:

The PAL self-test control word is a 48-bit value. This bit field is defined in Figure 11-9.

Add a Figure 11-9, showing 48 bits, bit 47 is named “cs” and bits 46-0 named “*test_control*”.

- *test_control* - This is an ordered implementation specific control word that allows the user control over the length and run-time of the processor self-tests. This control word is ordered from the longest running tests up to the shortest running tests with bit 0 controlling the longest running test.

  PAL may not implement all 47-bits of the test_control word. PAL communicates if a bit provides control by placing a zero in that bit. If a bit provides no control, PAL will place a one in it.

  PAL will have two sets of test control bits for the two phases of the processor self-test.

  PAL provides information about implemented *test_control* bits at the hand-off from PAL to SAL for the firmware recovery check. These *test_control* bits provide control for phase one of processor self-test. It also provides this information via the PAL procedure call

PAL_TEST_INFO for both the phase one and phase two processor tests depending on which information the caller is requesting.

PAL interprets these bits as input parameters on two occasions. The first time is when SAL passes control back to PAL after the firmware recovery check. The second time is when a call to PAL_TEST_PROC is made. When PAL interprets these bits it will only interpret implemented *test_control* bits and will ignore the values located in the un-implemented *test_control* bits.

PAL interprets the implemented bits such that if a bit contains a zero, this indicates to run the test. If a bit contains a one, this indicates to PAL to skip the test.

If the *cs* bit indicates that control is not available, the *test_control* bits will be ignored or generate an illegal argument in procedure calls if the caller sets these bits.

- *cs* - Control Support: This bit defines if an implementation supports control of the PAL self-tests via the self-test control word. If this bit is 0, the implementation does not support control of the processor self-tests via the self-test control word. If this bit is 1, the implementation does support control of the processor self-tests via the self-test control word.

  If control is not supported, GR37 will be ignored at the hand-off between SAL and PAL after the firmware recovery check and the PAL procedures related to the processor self-tests may return illegal arguments if a user tries to use the self-test control features.

On p. 11-31, Table 11-16, change the procedure name and description section.

FROM:

- PAL_MEM_FOR_TEST

TO:

- PAL_TEST_INFO

and

FROM:

Return the amount of memory needed for late processor self-test

TO:

Returns alignment and size requirements needed for the memory buffer passed to the PAL_TEST_PROC procedure as well as information on self-test control words for the processor self tests.

On p. 11-91, PAL_MEM_FOR_TEST procedure description:

a. Modify purpose statement to state the following:

   Returns the alignment and size requirements needed for the memory buffer passed to the PAL_TEST_PROC procedure as well as information on self-test control words for the processor self-tests.

b. Add a new Argument:

Argument	Description
test_phase	Unsigned integer that specifies which phase of the processor self-test information is being requested on. A value of 0 indicates the phase two of the processor self-test and a value of 1 indicates phase one of the processor self-test. All other values are reserved.

c. Add a new Return:

Return Value	Description
st_control	48-bit wide bit-field indicating if control of the processor self-tests is supported and which bits of the 'test_control' field are defined for use.

d. Modify the description section to the following:

PAL_TEST_INFO returns the size and alignment requirements for the memory buffer that is passed to the PAL_TEST_PROC procedure and returns information on the implementation of the self-test control word based on the *test_phase* input argument. Please see Section 11.2.3 for more information on the self-test control word.

When *test_phase* is equal to zero, information is returned about phase two of the processor self-test. These are the tests that require external memory to execute properly. When *test_phase* is equal to one, information is returned about phase one of the processor self-test. These are the tests that are normally run during PALE_RESET and do not require external memory to properly execute. When information is requested about phase one of the processor self-test a memory buffer and alignment argument will be returned as well since these tests may need to save and restore processor state to this memory buffer if executed from the PAL_TEST_PROC procedure.

On p. 11-102, PAL_TEST_PROC procedure description. Change the input arguments as follows:

FROM:

Argument	Description
index	Index of PAL_TEST_PROC within the list of PAL procedures.
test_address	64-bit physical address of main memory area to be used by processor self-test. The memory region passed must be cacheable, bit 63 must be zero.
test_size	Number of bytes of main memory to be used by processor self-test.
attributes	A 16-bit mask of memory attributes to be tested.

TO:

Argument	Description
index	Index of PAL_TEST_PROC within the list of PAL procedures.
test_address	64-bit physical address of main memory area to be used by processor self-test. The memory region passed must be cacheable, bit 63 must be zero.
test_info	Input argument specifying the size of the memory buffer passed and the phase of the processor self-test that should be run. See Figure 11-9.
test_params	Input argument specifying the self-test control word and the allowable memory attributes that can be used with the memory buffer. See Figure 11-9.

a. Change the description section to the following:

The PAL_TEST_PROC procedure will perform a phase of the processor self-tests as directed by the *test_info* and the *test_control* input parameters.

test_address points to a contiguous memory region to be used by PAL_TEST_PROC. This memory region must be aligned as specified by the alignment return value from PAL_TEST_INFO, otherwise this procedure will return with an invalid argument return value. The PAL_TEST_PROC routine requires that the memory has been initialized and that there are no known uncorrected errors in the allocated memory.

The *test_info* input parameter specifies the size of the memory buffer passed to the procedure and which phase of the processor self-test is requested to be run. (either phase one or phase two)

Add a figure that shows upper 8 bits as test_phase and lower 56 bits as buffer_size.

- *buffer_size* indicates the size in bytes of the memory buffer that is passed to this procedure. *buffer_size* must be greater than or equal in size to the *bytes_needed* return value of the PAL_TEST_INFO otherwise this procedure will return with an invalid argument return value.
- *test_phase* defines which phase of the processor self-tests are requested to be run. A value of zero indicates to run phase two of the processor self-tests. Phase two of the processor self-tests are ones that require external memory to execute correctly. A value of one indicates to run phase one of the processor self-tests. Phase one of the processor self-tests are tests run during PALE_RESET and do not depend on external memory to run correctly. When the caller requests to have phase one of the processor self-test run via this procedure call, a memory buffer may be needed to save and restore state as required by the PAL calling conventions. The procedure PAL_TEST_INFO informs the caller about the requirements of the memory buffer.

  The *test_params* input argument specifies which memory attributes are allowed to be used with the memory buffer passed to this procedure as well as the self-test control word. The self-test control word *test_control* controls the run-time and coverage of the processor self-test phase specified in the *test_phase* parameter.

Add a figure that shows upper 48 bits as test_control, next 8-bits as reserved and lower 8-bits as attributes.

- *attributes* specifies the memory attributes that are allowed to be used with the memory buffer passed to this procedure. The attributes parameter is a vector where each bit represents one of the virtual memory attributes defined by the architecture The bit field position corresponds to the numeric memory attribute encoding defined in Section 9.4, “Memory Attributes”. The caller is required to support the cacheable attribute for the memory buffer, otherwise an invalid argument will be returned.
- *test_control* is the self-test control word corresponding to the *test_phase* passed. This *test_control* directs the coverage and run-time of the processor self-tests specified by the *test_phase* input argument. Information about the self-test control word can be found on Section 11.2.3 and information on if this feature is implemented and the number of bits supported can be obtained by the PAL_TEST_INFO procedure call. If this feature is implemented by the processor, the caller can selectively skip parts of the processor self-test by setting *test_control* bits to a one. If a bit has a zero, this test will be run. The values in the unimplemented bits are ignored. If PAL_TEST_INFO indicated that the self-test control word is not implemented, this procedure will return with an invalid argument status if the caller sets any of the *test_control* bits.

  PAL_TEST_PROC will classify the processor after the self-test in one of four states: CATASTROPHIC FAILURE, FUNCTIONALLY RESTRICTED, PERFORMANCE RESTRICTED, or HEALTHY. These processor self-test states are described in Table 11-14 on p. 11-11. If PAL_TEST_PROC returns in the FUNCTIONALLY RESTRICTED or PERFORMANCE RESTRICTED states the self-test_status return value can provide additional information regarding the nature of the failure. In the case of a CATASTROPHIC FAILURE, the procedure does not return.

  The procedure will only perform memory accesses to the buffer passed to it using the memory attributes indicated in the attributes bit-field. The caller must ensure that the memory region passed to the procedure is in a coherent state.

  PAL_TEST_PROC may modify PSR bits or system registers as necessary to test the processor. These bits or registers must be restored upon exit from PAL_TEST_PROC with the exception of the translation caches, which are evicted as a result of testing. PAL_TEST_PROC is free to invalidate all cache contents. If the caller depends on the contents of the cache, they should be flushed before making this call. PAL_TEST_PROC requires that the RSE is set up properly to

handle spills and fills to a valid memory location if the contents of the register stack are needed. PAL_TEST_PROC requires that the memory buffer passed to it is not shared with other processors running this procedure in the system at the same time. PAL_TEST_PROC will use this memory region in a non-coherent manner.

8. Volume 2: VHPT walks and forward progress

On p. 4-5, Section 4.1.1.2, change the second bullet under “In order to ensure forward progress for IA-64 code, the following rules must be observed by the processor and software:”

TO:

The processor may occasionally invalidate the last TC entry inserted. The processor must guarantee visibility of the last inserted TC entry to all references while PSR.ic is zero. The processor must eventually guarantee visibility of the last inserted TC entry until an rfi sets PSR.ic to 1 and at least one instruction is executed with PSR.ic equal to 1, and completes without a fault or interrupt. The last inserted TC entry may be occasionally removed before this point, and software must be prepared to re-insert the TC entry on a subsequent fault. For example, eager or mandatory RSE activity, speculative VHPT walks, or other interruptions of the restart instructions may displace the software-inserted TC entry, but when software later re-inserts the same TC entry, the processor must eventually complete the restart instruction to ensure forward progress, even if that restart instruction takes other faults which must be handled before it can complete. If PSR.ic is set to 1 by instructions other than rfi, the processor does not guarantee forward progress.

On p. 4-5, Section 4.1.1.2, insert these two bullets below the bullet cited above:

- If software inserts an entry into the TLB with an overlapping entry (same or larger size) in the VHPT, and if the VHPT walker is enabled, forward progress is not guaranteed. See VHPT Searching, Section 4.1.5.2 on p. 4-15.
- Software may only make references to memory with physical addresses or with virtual addresses which are mapped with TRs, or to addresses mapped by the just-inserted translation, between the insertion of a TC entry, and the execution of the instruction with PSR.ic equal to 1 which is dependent on that entry for forward progress. Software may also make repeated attempts to execute the same instruction with PSR.ic equal to 1. If software makes any other memory references than these, the processor does not guarantee forward progress.

On p. 4-14, at the end of Section 4.1.5, add this paragraph:

If software needs to control the entries inserted into the TLB more explicitly, or programs the VHPT with differing mappings for the same virtual address range, it may need to take additional action to ensure forward progress. See VHPT searching, Section 4.1.5.2 on p. 4-15.

On p. 4-15, at the end of Section 4.1.5.2, add this paragraph:

VHPT walks may be done speculatively by the processor’s VHPT walker. Additionally, VHPT walks triggered by non-speculatively-executed instructions are not required to be done in program order. Therefore, if the walker is enabled and if the VHPT contains multiple entries that map the same virtual address range, software must set up these entries such that any of them can be used in the translation of any part of this virtual address range. Additionally, if software inserts a translation into the TLB which is needed for forward progress, and this translation has a smaller page size than the translation which would have been inserted on a VHPT walk for the same address, then software may need to disable the VHPT walker in order to ensure forward progress, since this inserted translation may be displaced by a VHPT walker before it can be used.

9. Revised Chapter 7 of Volume 4, Performance Monitoring Events (text included at end of this update)

Volume 4, Chapter 7, Performance Monitoring Events has been modified to include new content and reflect changes to enhance readability. The entire content of Chapter 7 are presented at the end of this update for convenience.

Note: The content in this specification update completely replaces Chapter 7 in Volume 4 of the *Intel® Itanium™ Software Developer's Manual*.

Specification Clarifications

1. Volume 2: Barrier clarification

1) On p. 13-20, Section 13.4.2, change Figure 13-5 by adding a "mf ;;" instruction at the very end of the code (after the loop) and fixing the comments to:

```
// The total shared variable is one less than the number of processors
// that wait at the barrier.
// The release shared variable indicates if the processor must wait at
// the barrier (initially, this variable is 0).
// local_sense is a per-processor local variable that indicates the
// "sense" of the barrier (initially, this variable is 0).

sr_barrier:
            fetchadd8.acq r1 = [count], 1// update counter
            ld8       r2 = [total]            // get number of procs - 1
            ld8       r3 = [local_sense] ;;   // get local “sense” variable
            xor       r3 = 1, r3              // local_sense =! local_sense
            cmp.eq    p1, p2 = r1, r2;;       // p1 => last proc to arrive
            st8       [local_sense] = r3      // save new value of local_sense
(p1)        st8       [count] = r0            // last resets count to 0
(p1)        st8.rel   [release] = r3 ;;       // last allows other to leave

wait_on_others:
(p2)        ld8          r1 = [release] ;;    // p2 => more procs to come
(p2)        cmp.ne.and   p0, p2 = r1, r3      // have all arrived yet?
(p2)        br.cond.sptk wait_on_others ;;    // nope, continue waiting

            // This mf prevents memory operations that follow the barrier code
            // from moving ahead of memory operations that precede the barrier
            // code
            mf ;;
```

2) On p. 13-21, add the following paragraph at the end of this section:

The `mf` instruction in Figure 13-5 is necessary only if the programmer wishes to ensure that memory operations performed before the barrier code are visible to memory operations performed by any processor after the barrier code.

2. Volume 2: Limited speculation clarifications

1) On p. 4-30, Section 4.4.6, change paragraph 3 (immediately after bulleted list) from:

Limited speculation is used for physical addressing to cached memory. ...or if the page is still enabled for prefetch through a speculative memory attribute.

to:

Limited speculation is used to improve performance when using physical addressing to cachable memory. Because the memory is physically addressed, the processor can have no expectation as to whether or not a given 4k-byte physical page exists until the page has been successfully accessed through a **non-speculative reference**. A non-speculative reference is an instruction or data reference made to the page by an in-order execution of the program. An instruction fetch (or data fetch) which meets this requirement, but which takes an Instruction

Debug (or Data Debug) fault or an External interrupt is still a non-speculative reference. Data-speculative references are considered non-speculative for this purpose. Control-speculative references are not allowed for limited-speculation pages and thus do not affect limited-speculation behavior.

Unless a limited-speculation page is **speculatively accessible**, only non-speculative references may be made to it. While a limited-speculation page is speculatively accessible, the processor may access it normally including the use of caching and **hardware-generated speculative references** to improve performance. Hardware-generated speculative references include non-demand instruction prefetches (including IA-32), data references by instructions which have not yet been determined to be required by an in-order execution of the program (due to potential exceptions on prior instructions or mispredictions on prior branches), hardware-generated data prefetch references, and eager RSE memory references. A limited-speculation page can be made speculatively accessible only after the successful completion of a non-speculative reference to the page. Once a limited-speculation page is speculatively accessible, the page can be be made **speculatively inaccessible** either explicitly by software (described in Section 4.4.11, “Memory Attribute Transition” on page 4-39) or implicitly for implementation-specific reasons.

2) On p. 4-31, Section 4.4.6, Table 4-12, add a new “Hardware-generated speculative references” column and footnotes to create:

Memory Attribute	Load (ld)[a]	Speculative Load (ld.s)[b]	Advanced Load (ld.a)	Speculative Advanced Load (ld.sa)	Hardware-generated Speculative References[c]
speculative	yes	yes	yes	yes	yes
non-speculative	yes	always fail	always fail	always fail	prohibited
limited speculation	yes	always fail	yes	always fail	limited[d]

a.Includes the faulting form of line prefetch (`lfetch.fault`).
b.Includes the non-faulting form of line prefetch (`lfetch`), which does not cause a cache fill if the memory attribute is non-speculative or limited speculation.
c.Hardware-generated speculative references include non-demand instruction prefetches (including IA-32), hardware-generated data prefetch references, and eager RSE memory references.
d.The processor may only issue hardware-generated speculative references to a 4K-byte physical page while the page is speculatively accessible.

3. Volume 3 and Volume 4: RCPPS/RCPSS and RSQRTPS/RSQRTSS clarifications

1) In Volume 3, Section 7.12, on pp. 7-79 and 7-80, RCPPS and RCPSS instructions,

a) change the second sentence of the Description section from:

The maximum error for this approximation is:

to:

The relative error for this approximation is Error, which satisfies:

b) change the last sentence of the Comment section from:

... and underflow results are always flushed to zero, with the sign of the operand.

to:

... and tiny results are always flushed to zero, with the sign of the operand.

Results are guaranteed not to be tiny, and therefore not flushed to zero, for input values x which satisfy

$|x| <= 1.11111111110100000000000_B \times 2^{125}$

For input values x which satisfy

$1.11111111110100000000001_B \times 2^{125}$ <= |x| <= $1.00000000000110000000000_B \times 2^{126}$ flush-to-zero might or might not occur, depending on the implementation (this interval contains 6144 + 3072 = 9216 single precision floating-point numbers).

Results are guaranteed to be tiny, and therefore flushed to zero, for input values x which satisfy

|x| >= $1.00000000000110000000001_B \times 2^{126}$

The decimal approximations of the single precision numbers that delimit the three intervals specified above, are as follows:

$1.11111111110100000000000_B \times 2^{125} \sim= 8.5039437 \times 10^{37}$
$1.11111111110100000000001_B \times 2^{125} \sim= 8.5039443 \times 10^{37}$
$1.00000000000110000000000_B \times 2^{126} \sim= 4.2550872 \times 10^{37}$
$1.00000000000110000000001_B \times 2^{126} \sim= 4.2550877 \times 10^{37}$

The hexadecimal representations of the single precision numbers that delimit the three intervals specified above, are as follows:

$1.11111111110100000000000_B \times 2^{125}$ = 0x7e7fe800

$1.11111111110100000000001_B \times 2^{125}$ = 0x7e7fe801

$1.00000000000110000000000_B \times 2^{126}$ = 0x7e800c00
$1.00000000000110000000001_B \times 2^{126}$ = 0x7e800c01

2) In Volume 3, on pp. 7-81 and 7-82, RSQRTPS and RSQRTSS instructions:

a) change the second sentence of the Description section from:

The maximum error for this approximation is:

to:

The relative error for this approximation is Error, which satisfies:

b) delete the following sentence from the Comment section:

“and underflow results are always flushed to zero, with the sign of the operand.”

3) In Volume 4, on p. 8-4, Section 8.12, change the last two sentences from:

The Pentium III processor implementation of one of these functions can have a maximum error of $1.5 * 10^{-12}$. The Itanium processor, however, will calculate these functions to a maximum error of $1.5 * 10^{-16}$.

to:

The Pentium III processor implementation of one of these functions can have a maximum relative error of $1.5 * 2^{-12}$. The Itanium processor, however, will calculate RCPPS/RCPSS functions with a maximum relative error of $2^{-17.75288} \sim= 1.1868 * 2^{-18}$ and the RSQRTPS/ RSQRTSS functions with a maximum relative error of $2^{-17.06412} \sim= 1.9130 * 2^{-18}$.

4. Volume 2: PAL code memory accesses and restrictions clarification

On p. 11-110, add a new Section 11.9 titled “PAL code memory accesses and restrictions” with the following text:

PAL issues load and store operations to memory in the following cases with the following memory attributes:

- during machine check/INIT handling to the min-state save area memory region registered with PAL using the UC memory attribute

- during the execution of PAL procedures to the memory buffer allocated by the caller of the procedure using the memory attribute of the address passed by the caller.
- PAL may also issue loads from the architected firmware address space and loads/stores for the registered min-state save area whenever it is executing a PAL procedure or handling PAL based interruptions (reset, MCA, INIT and PMI). PAL code may use either the UC or WBL memory attribute when accessing these areas.

PAL code will not send IPIs that require any special support from the platform.

5. Volume 2: PSP validity on INITs from PAL_MC_ERROR_INFO clarification

On p. 11-78, Section 11.8.3, in Table 11-40, change the description section for info_index of 1 to:

This *info_index* value will return the same processor state parameter that is passed at the PALE_CHECK exit state for a machine check event (provided a valid min-state save area has been registered) or will construct a processor state parameter for a corrected machine check events. This parameter describes the severity of the error and the validity of the processor state when the machine check or CMCI occurred. This procedure will not return a valid PSP for INIT events. The Processor State Parameter is described in ...

6. Volume 2: Non-speculative memory, misalignment and multiple accesses clarification

On p. 4-30 in Section 4.4.6, change the second bullet to:

- Will generate exactly one memory access for each aligned, non-speculative data reference. (Misaligned data references may cause multiple memory accesses, although these accesses are guaranteed to be non-overlapping - each byte will be accessed exactly once.)

7. Volume 2: PAL_A FIT entry, PAL_VM_TR_READ, PSP, and PAL_VERSION clarifications

1) On p. 11-8, add this note following the bullet text:

Note: The PAL_A FIT entry is not part of the FIT table checksum.

2) On p. 11-109, Section 11.8.3, change the following sentence in the Description section of PAL_VM_TR_READ from:

The information returned for the TR may have some invalid fields. The validity of the fields returned is signaled by the *TR_valid* return value.

to:

Some fields of the translation register returned may be invalid. The validity of these fields is indicated by the return argument *TR_valid*. If these fields are not valid, the caller should ignore the indicated fields when reading the translation register returned in *tr_buffer*.

3) On p. 11-15, Section 11.3.2.1, modify ‘me’ bit description of the processor state parameter from:

Distinct multiple errors have occurred, not multiple occurrences of a single correctable error. Software recovery is not possible. Some error information may have been lost.

to:

Distinct multiple errors have occurred, not multiple occurrences of a single error. Software recovery may be possible if error information has not been lost.

4) On p. 11-104, Section 11.8.3, modify PAL_VERSION Description Section bullets and last sentence to:

- *PAL_B_version* is a 16-bit binary coded decimal (BCD) number that provides identification information about the PAL_B firmware.
- *PAL_vendor* is an unsigned 8-bit integer indicating the vendor of the PAL code.
- *PAL_A_version* is a 16-bit binary coded decimal (BCD) number that provides identification information about the PAL_A firmware.

The version numbers selected for the PAL_A and PAL_B firmware is specific to the *PAL_vendor*. The version numbers selected will always have the property that later versions of firmware will have a higher number than earlier versions of firmware.

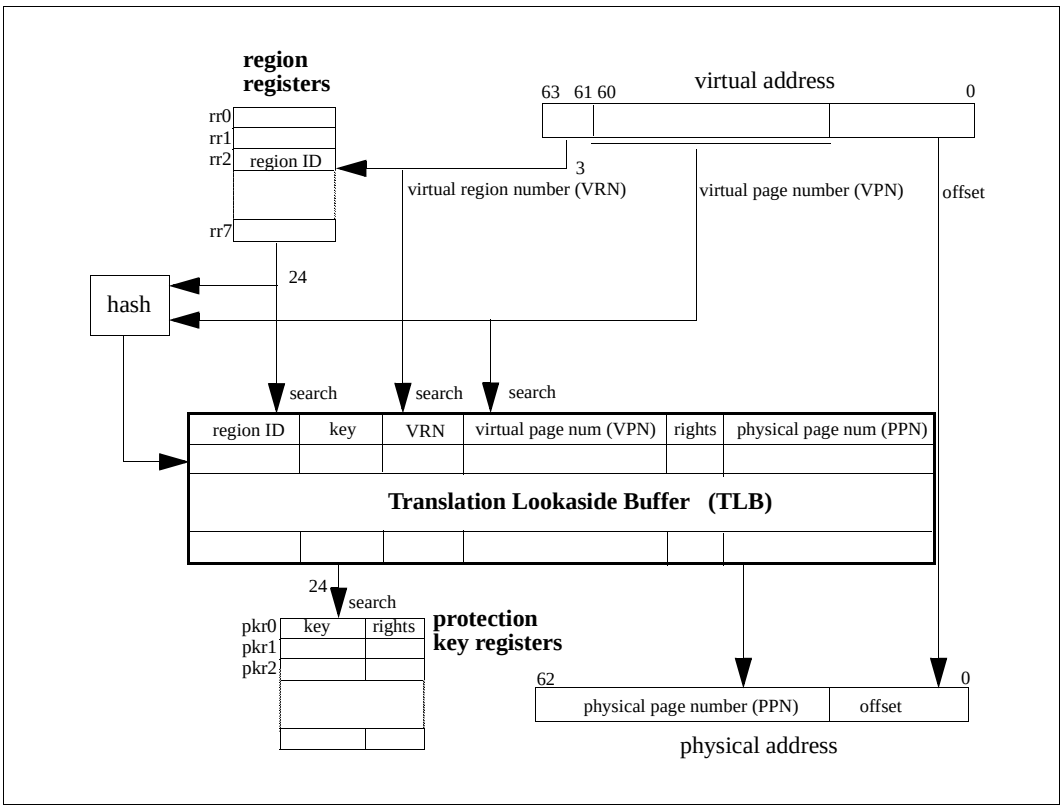
8. Volume 2: TLB searching clarification

1) On p. 4-3, Section 4.1, modify Figure 4-2:

a) change title to "Conceptual Virtual Address Translation for References"

b) add VRN field to the TLB entries, in-between the key and VPN fields

c) add an arrow from the VRN field in the virtual address to this new VRN field in the TLB entry and add the "search" label



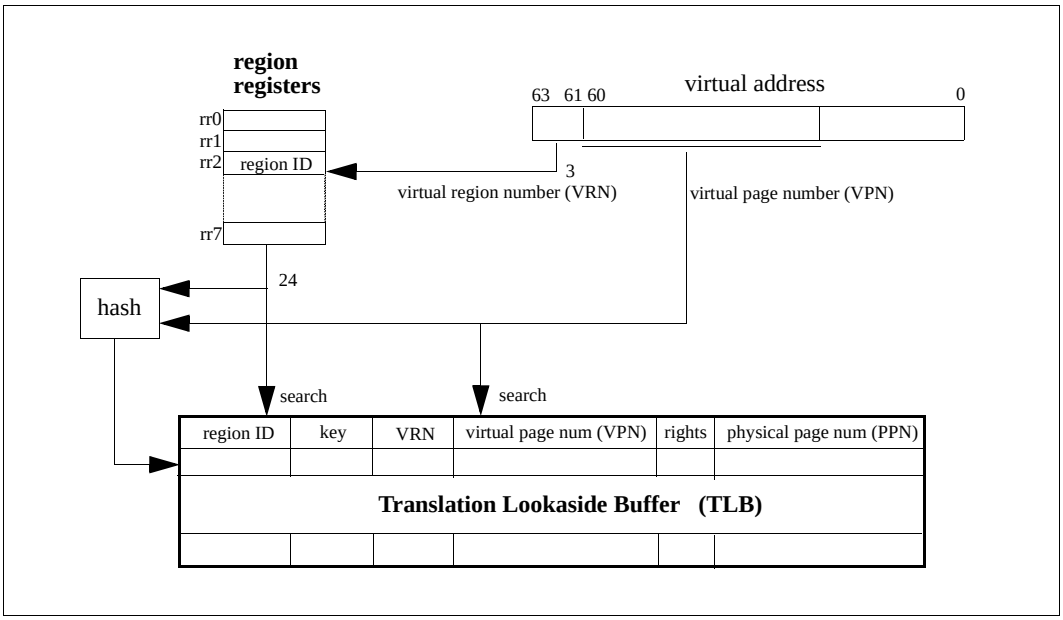
2) On p. 4-2, change the third paragraph to:

On a memory reference (any reference other than an insert or purge), the VRN bits select a Region Identifier (RID) from 1 of the 8 region registers, the TLB is then searched for a translation entry with a matching VPN and RID value. The VRN may optionally be used when

searching for a matching translation on memory references (references other than inserts and purges - see Section 4.1.1.4). If a matching translation entry is found, the entry's physical page number (PPN) is concatenated with the page offset bits to form the physical address. Matching translations are qualified by page-granular privilege level access right checks and optional protection domain checks by verifying the translation's key is contained within a set of protection key registers and read, write, execute permissions are granted.

3) On p. 4-6, Section 4.1.1.4,

a) add a new Figure 4-4 titled "Conceptual Virtual Address Searching for Inserts and Purges" after the second paragraph:



b) delete the first paragraph of the section:

"In addition to using the region identifier...."

c) add a new paragraph between the paragraph "Translation contained in the translation caches...." and the new Figure 4-4:

As described in Section 4.1, each TLB may contain a VRN field, and virtual address bits {63:61} may be used as part of the match for memory references (references other than inserts and purges). This binding of a translation to the VRN implies that a lookup of a given virtual address (region identifier/VPN pair) in either the translation cache or translation registers may result in a TLB miss if a memory reference is made through a different VRN (even if the region identifiers in the two region registers are identical). Some processor models may also omit the VRN field of the TLB, causing the TLB search on memory references to find an entry independent of VRN bits. However, all processor models are required, during translation cache purge and insert operations, to purge all possible translations matching the region identifier and virtual address regardless of the specified VRN.

---

9. Volume 2: PAL_CACHE_FLUSH clarifications

On pp. 11-33 and 11-34, Table 11-19:

- Remove footnote reference (a.) from the psr.ic bit
- Remove footnote (a.) described below the table.

On pp. 11-42 to 11-45, PAL_CACHE_FLUSH procedure changes:

a. Page 11-42, return status of 1 description. Replace with the following:

   Call has not completed flushing due to a pending interrupt.

b. Page 11-43, *int* bullet section. Remove all PMI references by replacing with the following:

   int - 1 bit field indicating if the processor will periodically poll for external interrupts while flushing the specified *cache_type(s)*.

   If this bit is a 0, unmasked external interrupts will not be polled. The processor will ignore all pending unmasked external interrupts until all cache lines in the specified *cache_type(s)* are flushed. Depending on the size of the processor's caches, bus bandwidth and implementation characteristics, flushing the caches can take a long period of time, possibly delaying interrupt response times and potentially causing I/O devices to fail.

   If this bit is a 1, external interrupts will be polled periodically and will exit the procedure if one is seen. If an unmasked external interrupt becomes pending, this procedure will return and allow the caller to service the interrupt before all cache lines in the specified *cache_type(s)* are flushed.

c. Page 11-44, third and fifth paragraphs. Remove a few sentences related to the psr.ic and psr.i settings:

   Old text:

   > This procedure makes one flush pass through all caches specified by *cache_type* and all sets and associativities within those caches. The specified *cache_type(s)* are ensured to be flushed only of cache lines resident in the caches prior to PAL_CACHE_FLUSH initially being called with the *progress_indicator* set to 0.
   >
   > This procedure must be called with PSR.i and PSR.ic set to zero to ensure external interrupts are not taken before this procedure begins to flush the cache(s) or while this procedure is terminating. PSR.i and PSR.ic must be zero regardless of the value of the *int* field.

   New text:

   > This procedure makes one flush pass through all caches specified by *cache_type* and all sets and associativities within those caches. The specified *cache_type(s)* are ensured to be flushed only of cache lines resident in the caches prior to PAL_CACHE_FLUSH initially being called with the *progress_indicator* set to 0.

d. Page 11-44, bullets listing contents of caches after this procedure exits. Add new bullets and remove references to PMI:

   Old text:

   > Due to the following conditions, software cannot assume that when this procedure completes the entire flush pass that the specified *cache_type(s)* are empty of all clean and/or modified cache lines.

— After an interruption, the flush pass resumes at the interruption point (specified by *progress_indicator*). Due to execution of the interrupt handlers during the flush pass, the specified caches may contain new and possibly modified cache lines in sections of the caches already flushed.

— Prior prefetches initiated before this procedure is called are disabled and flushed from the cache. However, if a speculative translation exists in either the ITLB or DTLB, speculative instruction or data prefetch operation could immediately reload a non-modified cache line after it was flushed. To ensure prefetches do not occur, software must remove all speculative translation before calling this routines. Alternatively, software can disable the TLBs by setting PSR.it, PSR.dt, and PSR.rt to 0.

— The specified caches may also contain PAL firmware code cache entries required to flush the cache.

New text:

Due to the following conditions, software cannot assume that when this procedure completes the entire flush pass that the specified *cache_type(s)* are empty of all clean and/or modified cache lines.

— After an interruption, the flush pass resumes at the interruption point (specified by *progress_indicator*). Due to execution of the interrupt handlers during the flush pass, the specified caches may contain new and possibly modified cache lines in sections of the caches already flushed. The caller specifies if this procedure should poll for interrupts via the *int* bit of the *operation* parameter.

— Prior prefetches initiated before this procedure is called are disabled and flushed from the cache as described above. However, if a speculative translation exists in either the ITLB or DTLB, speculative instruction or data prefetch operation could immediately reload a non-modified cache line after it was flushed. To ensure prefetches do not occur, software must remove all speculative translation before calling this routines. Alternatively, software can disable the TLBs by setting PSR.it, PSR.dt, and PSR.rt to 0.

— The specified caches may contain PAL firmware code cache entries required to flush the cache.

— The specified caches may contain PAL and SAL PMI code if this call was made with psr.ic = 1 and a PMI interrupt is seen during the execution of the call.

— The specified caches may contain SAL or OS machine check or INIT code if these handlers run in a cacheable mode and a machine check or INIT event is seen.

e. Page 11-45, remove PMI reference from the following paragraph:

Old text:

To ensure forward progress, PAL_CACHE_FLUSH advances through the cache flush sequence at least by one cache line before sampling for pending external interrupts or PMI. The amount of flushing that occurs before interrupts are polled will vary across implementations.

New text:

To ensure forward progress, PAL_CACHE_FLUSH advances through the cache flush sequence at least by one cache line before sampling for pending external interrupts. The amount of flushing that occurs before interrupts are polled will vary across implementations.

f. Page 11-45, *Status* and *vector* bullet descriptions. Remove PMI references by replacing bullets with the following:

Status

When the call returns a 1, it indicates that the call did not have any errors but is returning due to a pending unmasked external interrupt. To continue flushing the caches, the caller must call PAL_CACHE_FLUSH again with the value returned in the *progress_indicator* return value.

When the call returns a 0, it indicates that the call completed without any errors. All cache lines that were present in the cache (when the most recent call to PAL_CACHE_FLUSH with a *progress_indicator* of zero) are flushed and possibly invalidated. All intermediate calls must have used the proper *progress_indicator*, otherwise behavior is undefined.

vector - If the return status is 1 and this procedure exited due to a pending unmasked external interrupt, this field returns the interrupt vector number. The external interrupt will have been removed. The interrupt is considered to be “in-service” and software must service this interrupt for the specified vector and then issue EOI. If the return status is not 1, the values returned is undefined.

10. Volume 2: Non-speculative reference clarification

On p. 4-30, Section 4.4.6, just after the bullet list, add this sentence:

References that meet these requirements are termed non-speculative references. An instruction fetch which meets these requirements, but which takes an Instruction Debug fault or an External interrupt is still a non-speculative reference.

11. Volume 2: RID and Preferred Page Size usage clarification

On p. 4-11, Section 4.1.2, Table 4-5, add the following footnote to the “Description” column for “ps” field:

1.For more details on the usage of this field, please refer to Section 4.1.6 “VHPT Hashing”.

On p. 4-19, at the end of Section 4.1.6.2, replace item (1.) with:

1.Software must use only one preferred page size for each unique region identifier at any given time; otherwise, processor operation is undefined.

On p. 4-19, at the end of Section 4.1.6.2, add the following item (4.):

4.To reuse a region identifier with a different preferred page size, software must first ensure that the VHPT contains no insertable translations for that rid, purge all translations for that rid from all processors that may have used it, and then update the region register with the new preferred page size.

12. Volume 2: VHPT read atomicity clarification

On p. 4-17, Section 4.1.5.4, for this paragraph and bullet list:

Atomic updates of long-format VHPT entries may be ensured by software as follows:

- Before making multiple non-atomic updates to a VHPT entry in memory, software is required to set its ti bit to one.
- After making multiple non-atomic updates to a VHPT entry in memory, software may clear its ti bit to zero to re-enable tag matches.”

Change the first sentence to this:

For multi-processor systems, atomic updates of long-format VHPT entries may be ensured by software as follows:

Add this paragraph after the bullet list:

The updates to the VHPT entry in memory must be constrained to be observable only after the store that sets the it bit to one is observable. This can be accomplished with a mf instruction, or by performing the updates to the VHPT entry with release stores. Similarly, the clearing of the ti bit must be constrained to be observable only after all of the updates to the VHPT entry are observable. This can be accomplished with a mf instruction, or by performing the clear of the ti bit with a release store.

On p. 4-19, Section 4.1.7, for this paragraph:

VHPT walker references to the VHPT are performed at privilege level 0, regardless of the state of PSR.cpl. VHPT byte ordering is determined by the state of DCR.be. When DCR.be=1, VHPT walker references are performed using big-endian memory formats; otherwise, VHPT walker references are little-endian. The VHPT walker references the VHPT entry as a sequence of at least 8-byte atomic accesses. A long-format VHPT reference is matched against the data break-point registers as a 32-byte reference.

Delete the sentence:

The VHPT walker references the VHPT entry as a sequence of at least 8-byte atomic accesses.

On p. 4-20, Section 4.1.7, for this paragraph and bullet list:

The processor's VHPT walker is required to read and insert VHPT entries from memory atomically as follows:

- If the walker does not read an entry from memory atomically, and an update to part of the entry that is being installed is detected, the walker must abort the insert and deliver an Instruction/ Data TLB Miss.
- If the walker reads an entry from memory atomically, and an update to part of the entry that is being installed is detected, the walker must either abort the insert and deliver an Instruction/ Data TLB Miss, or ignore the update and install the complete old entry.
- If the purge address range of a TLB purge operation (ptc.l, ptc.e, local or remote ptc.g or ptc.ga, ptr.i,or ptr.d) overlaps the virtual address the walker is attempting to insert, then the walker must either abort the insert and deliver an Instruction/Data TLB Miss, or delay the purge operation until after the walker either completes the insertion or aborts the walk.

Replace with this:

The processor's VHPT walker is required to read and insert VHPT entries from memory atomically (an 8-byte atomic read-and-insert for short format, and a 32-byte atomic read-and-insert for long format).

On p. 4-32, Section 4.4.7, the first paragraph begins as follows:

As described in Section 4.4.7, read-after-write, write-after-write, and write-after-read dependencies to the same memory location (memory dependency) are performed in program order by the processor.

Add a footnote to this sentence that reads as follows:

Although VHPT walks are performed somewhat asynchronously with respect to program execution, each walker VHPT read appears as though it were performed atomically, at some single point in the program order.

13. Volume 2: IIP and WC flush clarifications

On p. 4-29, Section 4.4.5, replace the 4th paragraph:

Any IA-64 release operation (regardless of whether it references a page with a coalescing memory attribute), or any IA-64 fence type instruction, forces write-coalesced data to become visible prior to the instruction itself becoming visible. (See Table 4-14 for a list of release and fence instruction.) Any IA-32 serializing instruction, or access to an uncached memory type, forces write-coalesced data to become visible prior to itself becoming visible.

WITH:

Any IA-64 release operation (regardless of whether it references a page with a coalescing memory attribute), or any IA-64 fence type instruction, forces write-coalesced data to be flushed and made visible prior to the instruction itself becoming visible (See Table 4-14 for a list of release and fence instruction.) Any IA-32 serializing instruction, or access to an uncached memory type, forces write-coalesced data to become flushed and made visible prior to itself becoming visible.

On p. 8-36, Section 8.3, parameters. Replace the following:

IIP, IPSR, IIPA, IFS - are defined; refer to p. 8-1 for a detailed description.

WITH:

IIP, IPSR, IIPA, IFS - are defined; refer to p. 8-1 for a detailed description.

Note: Please see Section 3.3.5.3 on page 3-18 for a further clarification of the IIP value for an unimplemented instruction address trap.

14. Volume 1: Faults in ld.c that hits ALAT clarification

On p. 4-19, Section 4.4.5.3.1, Item 2 in the middle of the page:

1. If the implementation chooses to leave the target register unchanged and one or more exception conditions related to the data access or translation of the check load occurs, the implementation may choose to either raise the highest-priority of these faults or ignore them all and continue execution. The faults that can be ignored are those related to data access and translation (Data Nested TLB fault, Alternate Data TLB fault, VHPT Data fault, Data TLB fault, Data Page Not Present fault, Data NaT Page Consumption Fault, Data Key Miss fault, Data Key Permission fault, Data Access Rights fault, Data Dirty bit fault, Data Access Bit fault, Data Debug fault, Unaligned Data Reference fault, Unsupported Data Reference fault.

15. Volume 2: IA-32 IBR/DBR match clarification

On p. 7-3, Table 7-1, change the “addr” field description as follows:

FROM:

For IA-32 instruction set references, IBR{31:0} are used in the match. For IA-32 memory references, addr{63:32} must be zero to match.

TO:

For IA-32 instruction references, IBR.addr{31:0} are used in the match and IBR.addr{63:32} must be zero to match.

On p. 7-3, Table 7-1, change the “mask” field description

FROM:

For IA-32 memory references, mask{63:32} are ignored

TO:

For IA-32 instruction references, IBR.mask{55:32} are ignored

On p. 7-5, remove this sentence from the last bullet:

The upper 32-bits of DBR addr field must be zero to detect IA-32 data memory references.

16. Volume 3: IA-32 CPUID clarification

On p. 5-71, change

FROM:

```
        BREAK;
ESAC;
```

TO:

```
        BREAK;
        DEFAULT: (* EAX > highest value recognized by CPUID *)
                EAX <- Reserved, undefined;
                EBX <- Reserved, undefined;
                ECX <- Reserved, undefined;
                EDX <- Reserved, undefined;
        BREAK;
ESAC;
```

17. Volume 2: Table A-5 class pr-writers-int

In Table A-5 on p. A-22, for the class pr-writers-int, add pr-and-writers and pr-or-writers to the Events/Instructions column.

18. Volume 2: Section 4.4.6.1 PAL_MC_DRAIN procedure only causes cache line writeback transactions to be forced onto the bus, and does not guarantee that they reached main memory

Change the last paragraph of Section 4.4.6.1on p. 4-32 to:

To further guarantee that any cache lines containing addresses belonging to page [X] have been evicted from all caches in the coherence domain and *forced onto the bus*, software must perform a PAL_MC_DRAIN operation on all processors in the coherence domain (via the IPI mechanism) after executing the above sequence. Note that this operation does not ensure that the cache lines have been written back to memory.

(In the paragraph above, the words “written back to memory” have been changed to “forced onto the bus” and a new sentence was added at the end.)

Documentation Changes

1. Volume 1, Volume 2, Volume 3, and Volume 4: IA-32 related documentation changes

1) In Volume 1, on p. 6-18, Section 6.2.5.4 , bullets describing the ES-bit. Remove the SF-bit reference from these bullets.

2) In Volume 1, clarify which fields are checked when IA-32 ARs are written when in IA-64 mode.

a) On p. 6-9, Section 6.2.3, change the first paragraph from:

"... IA-64 software can also directly load descriptor registers provided they are properly unscrambled by software. For a complete definition ..."

to:

“... Itanium-based software can also directly load descriptor registers provided they are properly unscrambled by software. When Itanium-based software loads these registers, no data integrity checks are performed at that time if illegal values are loaded in any fields. For a complete definition ...”

b) On p. 6-15, Section 6.2.4, change the first paragraph from:

"... None of the arithmetic or system flags affect IA-64 instruction execution. See ..."

to:

"... None of the arithmetic or system flags affect Itanium instruction execution. When Itanium-based software loads this application register (AR24), a Reserved Register/Field fault will be raised if a non-zero value is written into bits listed as reserved. See ..."

c) On p. 6-18, Section 6.2.5.3, change the last paragraph before Section 6.2.5.4 from:

"Software must ensure that FCR and FSR are properly loaded for IA-32 numeric execution before entering the IA-32 instruction set."

to:

"Software must ensure that FCR and FSR are properly loaded for IA-32 numeric execution before entering the IA-32 instruction set. When Itanium-based software loads these application registers (AR21 and AR28), a Reserved Register/Field fault will be raised if a non-zero value is written to bits listed as reserved. No field encoding values will be verified when these registers are written."

d. On p. 6-20, Section 6.2.5.4, change the last paragraph from:

"FSR, FDR, and FIR must be preserved across a context switch to generate and accurately report numeric exceptions."

to:

"When Itanium-based software loads these application registers (AR29 and AR30), a Reserved Register/Field fault will be raised if a non-zero value is written to bits listed as reserved. No field encoding values will be verified when these registers are written. FSR, FDR, and FIR must be preserved across a context switch to generate and accurately report numeric exceptions."

3) In Volume 2, on p. 10-3. Section 10.3, change the last sentence of the first paragraph from:

"IA-64 software can also directly load descriptor registers provided they are properly unscrambled by software. For a complete definition ..."

to:

"Itanium-based software can also directly load descriptor registers provided they are properly unscrambled by software. When Itanium-based software loads these registers, no data integrity checks are performed at that time if illegal values are loaded in any fields. For a complete definition ..."

4) In Volume 2, on p.10-5, add a new paragraph at the end of Section 10.3.2 that reads:

When Itanium-based software loads this application register (AR24), a Reserved Register/ Field fault will be raised if a non-zero value is written into bits listed as reserved.

5) In Volume 2, on p. 10-7, Section 10.3.3.1, change the end of first paragraph from:

"CFLG is readable by IA-64 code at all privilege levels but can only be written and privilege level 0, otherwise a Privileged Register fault is generated."

to:

"CFLG is readable by Itanium-based code at all privilege levels but can only be written at privilege level 0, otherwise a Privileged Register fault is generated. When Itanium-based software loads this application register (AR24), a Reserved Register/Field fault will be raised if a non-zero value is written into bits listed as reserved."

6) In Volume 2, on pp. 10-35 and 10-36, change Section 10.10.1 text to:

Within the Itanium System Environment, the following bus transactions are initiated:

- INTA - Interrupt Acknowledge - emitted by the operating system (via a read to the INTA byte in the processor's Interrupt Block) to acquire the interrupt vector number from an external interrupt controller.
- HALT - Emitted when the processor has entered the halt state due to the operating system /platform firmware calling PAL_HALT or PAL_HALT_LIGHT.
- SHUTDOWN - Emitted when the processor has entered the shutdown state. This can only be generated when the processor has entered into the IA-32 System Environment by calling PAL_ENTER_IA_32_ENV procedure call.
- STPACK - Stop Acknowledge. Emitted by calling an implementation specific PAL firmware procedure. See the processor specific firmware guide for more information.
- FLUSH - Emitted when the WBINVD or INVD instruction is executed when running in the IA-32 System Environment entered by calling PAL_ENTER_IA_32_ENV procedure call. Indicates that external caches (if any) should be invalidated.
- SYNC - Emitted when the WBINVD instruction is executed when running in the IA-32 System Environment entered by calling PAL_ENTER_IA_32_ENV procedure call. Indicates that external caches (if any) should copy all modified cache lines back to main memory.

7) In Volume 3, Section 5.3, on p. 5-71, CPUID instruction page, add a new paragraph after bullet items:

Please see the programmer's guide for further information on how to decode return values for the processor's internal caches and TLBs.

8) In Volume 4, on p. 8-2, add a new Section 8.4 titled "CPUID Instruction Return Values for Caches and TLBs of the Itanium Processor" and the following text and table:

The following table provides information on how to decode return values of the CPUID instruction for the Itanium processor's internal caches and TLBs.

Table 8-1. Encoding of Cache and TLB Return Values for the Itanium Processor

Return Value	Cache or TLB Description
0x10	L0D: 16K 4-way set associative 32 bytes line
0x15	L0I: 16K 4-way set associative 32 bytes line
0x1A	L1: 96K on die 6-way set associative 64 byte line
0x88	L2: 2M 4-way set associative 64 bytes line
0x89	L2: 4M 4-way set associative 64 bytes line
0x8A	L2: 8M 4-way set associative 64 bytes line
0x90	ITLB: 4K to 256M pages, fully associative, 64 entries
0x96	DTLB0: 4K to 256M pages, fully associative, 32 entries
0x9B	DTLB1: 4K to 256M pages, fully associative, 96 entries

When the input value in register EAX is 2, the Itanium processor returns information about the processor's internal caches and TLBs in the EAX, EBX, ECX, and EDX registers. The following table describes the values returned.

Table 8-2. EAX, EBX, ECX, and EDX Return Values for the Itanium Processor

Register	Return Value (from MSB to LSB)
EAX	0x00, 0x15, 0x10, 0x01
EBX	0x00, 0x00, <L2>, 0x1A (<L2> is either 0x88, 0x89)
ECX	0x00, 0x9B, 0x96, 0x90
EDX	0x80, 0x00, 0x00, 0x00

2. Volume 1: Load instructions change

On p. 4-11, Section 4.4.1, replace the 3rd paragraph from:

For floating-point loads, five access sizes are defined: single precision (4 bytes), double precision (8 bytes), double-extended precision (10 bytes), single precision pair (8 bytes), and double precision pair (16 bytes). The value(s) loaded from memory are converted into floating-point register format (see "Memory Access Instructions" on page 5-7 for details). The floating-point load pair instructions load two adjacent single or double precision numbers into two independent floating-point registers (see the `ldfp[s/d]` instruction description for restrictions on target register specifiers). The floating-point load pair instructions cannot specify base register update.

to:

For floating-point loads, the following access sizes are defined: single precision (4 bytes), double precision (8 bytes), double-extended precision (10 bytes), and integer/parallel FP (8 bytes). The value(s) loaded from memory are converted into floating-point register format (see "Memory Access Instructions" on page 5-7 for details).

The floating-point load pair instructions load two adjacent single precision (4 bytes each), double precision (8 bytes each), or integer/parallel FP (8 bytes each) numbers into two independent floating-point registers (see the `ldfp` instruction description for restrictions on target register specifiers). Floating-point load pair instructions can specify base register update, but only by an immediate value equal to double the data size.

3. Volume 3: tak, tpa change

1) On p. 2-220, Section 2.2, replace the description from:

"When PSR.dt is 0, `tak` searches the DTLB only, because the VHPT walker is disabled..."

to:

"When PSR.dt is 0, only the DTLB is searched, because the VHPT walker is disabled..."

2) On p. 2-226, Section 2.2, replace the description from:

"When PSR.dt is 0, `tak` searches the DTLB only, because the VHPT walker is disabled. If no matching present translation is found in the DTLB, an Alternate Data TLB fault is raised."

to:

"When PSR.dt is 0, only the DTLB is searched, because the VHPT walked is disabled. If no matching present translation is found in the DTLB, an Alternate Data TLB fault is raised if psr.ic is one or a Data Nested TLB fault is raised if psr.ic is zero."

4. Volume 2: IPSR.ri and ISR.ei changes

1) On p. 3-10, Section 3.3.2, Table 3-2, replace description of the 'ri' bit with:

Restart Instruction - Set on an interruption, indicating the next instruction in the bundle to be executed. When the next instruction is the L+X instruction of an MLX, this field is set to the value 1.

When restarting instructions with `rfi`, this field specifies which instruction(s) in the bundle are restarted. The specified and subsequent instructions are restarted, all instructions prior to the restart point are ignored.

0 - restart execution at instruction slot 0

1 - restart execution at instruction slot 1

2 - restart execution at instruction slot 2

3 - reserved

Except at an interruption and for the first restart instruction following an `rfi`, the value of this field is undefined. This field is set to 0 after any interruption from the IA-32 instruction set and is ignored when IA-32 instructions are restarted.

2) On p. 3-16, Section 3.3.5.1, delete the last sentence "IPSR.ri is set to 0, after any interruption from the IA-32 instruction set."

3) On p. 3-17, Section 3.3.5.2, revise the description of 'ei':

a) replace the binary values "00", "01", and "10" with the decimal equivalents, "0", "1", "2".

b) replace the sentence "For interruptions taken out of the IA-32 instruction set, ISR.ei is always 0." with "This field is always 0 for interruptions taken in the IA-32 instruction set."

4) On p. 5-7, Section 5.5,

a) change the third sentence of the first bullet of the "If PSR.ic is 1" case of step 1 of interruption handling from:

For all other interruptions, the value written to IPSR.ri is the instruction slot on which the interruption occurred.

to:

For all other interruptions, the value written to IPSR.ri is the instruction slot on which the interruption occurred (1 for interruptions on the L+X instruction of an MLX).

b) change the second sentence of step 2 of the "If PSR.ic is 1" case of interruption handling from:

The IA-64 instruction slot which caused the interruption is saved in ISR.ei.

to:

The instruction slot which caused the interruption is saved in ISR.ei (2 for traps, 1 for other interruptions, on the L+X instruction of an MLX).

5) On p. 8-5, Section 8.3, Table, 8-2, change footnote a to:

ISR.ei is equal to IPSR.ri for all faults and external interrupts (1 for faults and interrupts on the L+X instruction of an MLX). For traps, ISR.ei points at the excepting instruction (2 for traps on the L+X instruction of an MLX).

6) On p. 2-197, Section 2.2, modify the rfi description from:

If the target is a bundle containing a movl instruction and if this instruction sets PSR.ri to 2, then an Illegal Operation fault will be taken on the target bundle.

to:

If this instruction sets PSR.ri to 2 and the target is an MLX bundle, then an Illegal Operation fault will be taken on the target bundle.

5. Volume 4: Miscellaneous Chapter 7 “Performance Monitoring Events” typographical errors (text included at end of this update)

Volume 4, Chapter 7, Performance Monitoring Events has been modified to include new content and reflect changes to enhance readability. The entire content of Chapter 7 are presented at the end of this update for convenience.

Note: The content in this specification update completely replaces Chapter 7 in Volume 4 of the *Intel® Itanium™ Software Developer's Manual*.

1) On p. 7-47, Table 7-15 “Cache Performance Ratios”, replace “L3_DATA_REFERENCES.d” with “L2_INST_REFERENCES.d” for L1I Miss Ratio and L2 Instruction Ratio.

2) On p. 7-49, Table 7-17 “L1 Data Cache Monitors”, replace “L1I_PREFETCH_READS” with “L2_DATA_REFERENCES.ALL”.

3) On p. 7-71, BRANCH_TAKEN_SLOT, replace second “Definition” with “Qualification”.

---

6. Volume 4: IBR and DBR addressing typographical errors

On p. 6-19, Section 6.2.4, in the middle of the page there are several equations. Replace all IBR_i.addr to $IBR_{[2*i]}$.addr:

Old text:

$IBRmatch_i$ = match(IP,IBR_i.addr,$IBR_{[2*i]+1}$.mask,$IBR_{[2*i]+1}$.plm

New text:

$IBRmatch_i$ = match(IP,$IBR_{[2*i]}$.addr,$IBR_{[2*i]+1}$.mask,$IBR_{[2*i]+1}$.plm

*Note - () are changed to [], to make the equations consistent with others.

Old text:

$IBRmatch_i$ = ($IBR_{[2*i]+1}$.plm[PSR.cpl])

and ($AND_{b=50..0}$((IBR_i .addr{b} and $IBR_{[2*i]+1}$.mask{b}) = (IP{b} and $IBR_{[2*i]+1}$.mask{b})))

and ($AND_{b=55..51}$((IBR_i .addr{b} and $IBR_{[2*i]+1}$.mask{b}) = (IP{50} and $IBR_{[2*i]+1}$.mask{b})))

and ($AND_{b=60..56}(IBR_i.addr\{b\} = IP\{50\})$)

and ($AND_{b=63..61}(IBR_i.addr\{b\} = IP\{b\})$)

New text:

$IBRmatch_i$ = ($IBR_{[2*i]+1}.plm[PSR.cpl]$)

and ($AND_{b=50..0}((IBR_{[2*i]}.addr\{b\}$ and $IBR_{[2*i]+1}.mask\{b\}) = (IP\{b\}$ and $IBR_{[2*i]+1}.mask\{b\})))$

and ($AND_{b=55..51}((IBR_{[2*i]}.addr\{b\}$ and $IBR_{[2*i]+1}.mask\{b\}) = (IP\{50\}$ and $IBR_{[2*i]+1}.mask\{b\})))$

and ($AND_{b=60..56}(IBR_{[2*i]}.addr\{b\} = IP\{50\})$)

and ($AND_{b=63..61}(IBR_{[2*i]}.addr\{b\} = IP\{b\})$)

On p. 6-21, Section 6.2.6, at the bottom of the page there are several equations. Replace all $DBR_i.addr$ to $DBR_{[2*i]}.addr$:

Old text:

$DBRRangeMatch_i$ =

($AND_{b=50..0}((DBR_i.addr\{b\}$ and $DBR_{[2*i]+1}.mask\{b\}) = (addr\{b\}$ and $DBR_{[2*i]+1}.mask\{b\})))$

and ($AND_{b=55..51}((DBR_i.addr\{b\}$ and $DBR_{[2*i]+1}.mask\{b\}) = (addr\{50\}$ and $DBR_{[2*i]+1}.mask\{b\})))$

and ($AND_{b=60..56}(DBR_i.addr\{b\} = addr\{50\})$)

and ($AND_{b=63..61}(DBR_i.addr\{b\} = addr\{b\})$)

New text:

$DBRRangeMatch_i$=

($AND_{b=50..0}((DBR_{[2*i]}.addr\{b\}$ and $DBR_{[2*i]+1}.mask\{b\}) = (addr\{b\}$ and $DBR_{[2*i]+1}.mask\{b\})))$

and ($AND_{b=55..51}((DBR_{[2*i]}.addr\{b\}$ and $DBR_{[2*i]+1}.mask\{b\}) = (addr\{50\}$ and $DBR_{[2*i]+1}.mask\{b\})))$

and ($AND_{b=60..56}(DBR_{[2*i]}.addr\{b\} = addr\{50\})$)

and ($AND_{b=63..61}(DBR_{[2*i]}.addr\{b\} = addr\{b\})$)

7. Volume 3: Figure changes for extract, deposit, and alloc instructions

On p. 2-40, the extract instruction, change the text just before Figure 2-6

FROM:

The operation of extr t = r, 7, 50 is illustrated...

TO:

The operation of extr $r_1 = r_3$, 7, 50 is illustrated...

On p. 2-40, change the labels of the register values in Figure 2-6

FROM:

GR r , to: GR r_3

AND FROM:

GR t , to: GR r_1

On p. 2-37, the deposit instruction, change the text just before Figure 2-5

FROM:

The operation of dep t = s, r, 36, 16 is illustrated...

TO:

The operation of dep r_1 = r_2, r_3, 36, 16 is illustrated...

On p. 2-37, change the labels of the register values in Figure 2-5

FROM:

GR r , to: GR r_3

AND FROM:

GR s , to: GR r_2

AND FROM:

GR t , to: GR r_1

On p. 2-37, after Figure 2-5, add this new paragraph:

The operation of dep.z r_1 = r_2, 36, 16 is illustrated in Figure XXX

Copy Figure 2-5 and the ParaAnchor paragraph that contains it.

Paste the copy just below the new paragraph.

In the copy, remove the label and register for GR r_3 and for the two lines from that register pointing to the result register.

Add a “0” in the left and right portions of the GR r_1 register, to show that these bits get 0. (For an example of a “0” in the target register in a figure, see the addp instruction, p. 2-4.)

On p. 2-5, the alloc instruction, Figure 2-2, add a double-arrow line below the sol line.

The left end of this new line should be aligned with the left end of the sol line.

The new line should be about half as long as the sol line.

Below the new line, create the text “sor”, centered under the new line.

8. Volume 2: RSE and PMC typographical errors

On p. 6-6, Section 6.4, Table 6-2, change the table entry for the CFM row, br.call column:

FROM:

CFM.sof = CFM.sol

TO:

CFM.sof -= CFM.sol

On pp. 7-7 and 7-8, Section 7.2.1, change “PMD” to “PMC” in three places:

The title of Figure 7-5

FROM:

...(PMC[4]..PMD[p])

TO:

...(PMC[4]..PMC[p])

In Figure 7-5, the left-hand side of the figure is a cell containing:

...(PMC[4]..PMD[p])”

TO:

...(PMC[4]..PMC[p])

In Table 7-4, change the title

FROM:

...(PMC[4]..PMD[p])

TO:

...(PMC[4]..PMC[p])

9. Volume 2: DV table typographical error

On p. A-22, Appendix A, Section A.4, Table A-5 (Instruction Classes), add instruction 'setf' into list for class fp-writers.

10. Volume 2: ISR figure and wording changes

On p. 8-5 of Volume 2, change the Unsupported Data Reference fault row in Table 8-2 so that the r column reads “r”, and the w column reads “w”.

On p. 8-26 of Volume 2, change the second ISR figure on the General Exception vector page to show sp as always 0.

On p. 8-33 of Volume 2, change the ISR figure on the Unsupported Data Reference vector page so that bit 34 reads “r”, and bit 33 reads “w”.

On p. 8-33 of Volume 2, add the following statement below the ISR figure on the Unsupported Data Reference vector page:

For ldfe and stfe instructions, the processor may optionally set both ISR.r and ISR.w to 1, although this is not recommended.

11. Volume 2: ISR figure change on the Lower-privilege Transfer Trap vector page (new p. 8-36 of Volume 2)

Name **Lower-privilege Transfer Trap vector (0x5e00)**

Cause Two trapping conditions transfer control to this vector:

- An attempt is made to execute an instruction at an unimplemented address, resulting in an Unimplemented Instruction Address trap. See “Unimplemented Address Bits” on p. 4-24.

- The PSR.lp bit is 1, and a branch lowers the privilege level.

IA-32 instructions can not raise this trap.

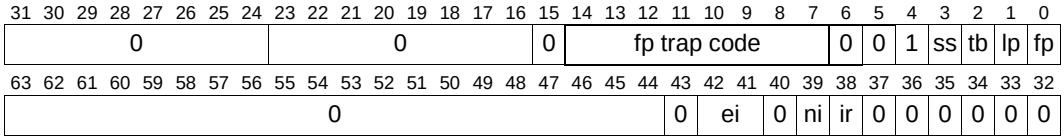
Interruptions on this vector:

Unimplemented Instruction Address trap
Lower-privilege Transfer trap

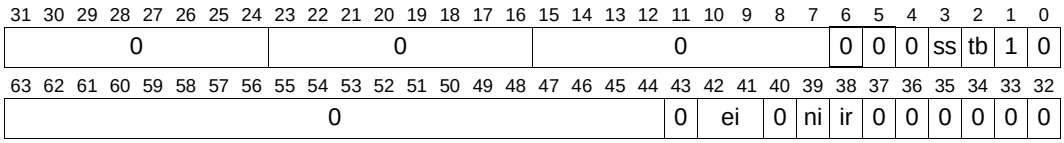
Parameters IIP, IPSR, IIPA, IFS – are defined; refer to p. 8-1 for a detailed description.

ISR – The ISR.ei bits are set to indicate which instruction caused the exception. The ISR.code contains a bit vector (see Table 8-3 on p. 8-5) for all traps which occurred in the just-executed instruction. The defined ISR bits are specified below.

If the trap is due to an Unimplemented Instruction Address trap:



If the trap is due to a Lower-Privilege Transfer trap:



Notes The Unimplemented Instruction Address trap can be the result of an inline instruction fetch, a taken or not-taken branch or an `rfi`. The lower privilege transfer trap is only taken on a branch demotion, and not an `rfi` return.

12. Volume 4: Figure 6-17 and Figure 6-19 bit typo

In Figure 6-17, Instruction Event Address Configuration Register (PMC[10]) and Figure 6-19, Data Event Address Configuration Register (PMC[11]) on p. 6-22 and p. 6-25 respectively, the second ‘ignored’ field (bits 8-15) from the right is showing 7 bits where it should be 8 bits.

13. Volume 4: Chapter 6 the eight stall and flush reasons

On p. 6-19, replace the text before Section 6.1.2, Profiling with the following:

The Itanium processor cycle accounting monitors account for all major single and multi-cycle stall and flush conditions. Overlapping stall and flush conditions are prioritized in reverse pipeline order (i.e. delays that occur later in the pipe and that overlap with earlier stage delays are reported as being caused later in the pipeline). The eight stall and flush reasons are prioritized in the following order:

1.Back-end Flush Cycles: cycles lost due to branch mispredictions, ALAT flushes, serialization flushes, failed control speculation flushes, MMU-IEU bypasses and other exceptions.

2.Data Access Cycles: cycles lost when instructions stall waiting for their source operands from the memory subsystem, and when memory flushes arise (L1D way mispredictions, DTC flushes).

3.Scoreboard Dependency Cycles: cycles lost when instructions stall waiting for their source operands from non-load instructions; this includes FP-related flushes.

4.RSE Active Cycles: stalls due to register stack spills to and fills from the backing store in memory.

5.Issue Limit Cycles: dispersal breaks due to stops, port over-subscription or asymmetries.

6.Instruction Access Cycles: instruction fetch stalls due to L1I or ITLB misses.

7.Taken Branch Cycles: bubbles incurred on correct taken branch predictions.

8.Unstalled Pipeline Cycles: cycles due to the inherent execution of the program.

Four of the eight categories (1, 2, 3, 6) are directly measurable as Itanium processor events. The other four categories (4, 5, 7, 8) are not measured directly. Instead, four combined categories are available as the Itanium processor events: pipeline flush cycles (1+7), memory cycles (2+4), dependency cycles (3+5), and unstalled back-end cycles (6+8). For details, refer to Section 7.4, “Cycle Accounting Events”.

This page intentionally left blank.

Performance Monitor Events 7

This chapter describes the architectural and microarchitectural events on the Itanium processor whose occurrences are countable through the performance monitoring mechanisms described earlier in Chapter 6. The earlier sections of this chapter aim to provide a high-level view of the event list, grouping logically related events together. Computation (either directly by a counter in hardware, or indirectly as a “derived” event) of common performance metrics is also discussed. Each directly measurable event is then described in greater detail in the alphabetized list of all processor events in Section 7.8, “Performance Monitor Event List”.

7.1 Categorization of Events

Performance related events are grouped into the following categories:

- Basic Events: clock cycles, retired instructions (Section 7.2)
- Instruction Execution: instruction decode, issue and execution, data and control speculation, and memory operations (Section 7.3)
- Cycle Accounting Events: stall and execution cycle breakdowns (Section 7.4)
- Branch Events: branch prediction (Section 7.5)
- Memory Hierarchy: instruction and data caches (Section 7.6)
- System Events: operating system monitors, instruction and data TLBs (Section 7.7)

Each section listed above includes a table of all events (both directly measured and derived) in that category. Directly measurable events often use the PMC.umask field (See Table 6-7 in Chapter 6) to measure a certain variant of the event in question. Symbolic event names for such events (e.g. ALAT_REPLACEMENT.ALL) include a period to indicate use of the umask, specified by 4 bits in the detailed event description (x’s are for don’t-cares). Derived events are computable from directly measured events and include a “.d” suffix in their symbolic event names. Formulas to compute relevant derived events also appear in each section. Derived events are not, however, discussed in the systematic event listing in Section 7.8.

The tables in the subsequent sections define events by specifying three attributes: symbolic event name, a brief event title and a reference to the detailed event description page. Derived events are not listed in the detailed event description pages and hence lack the appropriate reference.

7.2 Basic Events

Table 7-1 summarizes four basic execution monitors. The CPU_CYCLES event can be used to break out separate or combined IA-64 or IA-32 cycle counts (by constraining the PMC/PMD based on the currently executing instruction set). The IA-64 retired instruction count (IA64_INST_RETIRED) includes predicated true and false instructions, and nops, but excludes RSE operations.

Table 7-1. IA-64 and IA-32 Instruction Set Execution and Retirement Monitors

Execution Monitors	Title	Page
CPU_CYCLES	CPU Cycles	7-41
IA64_INST_RETIRED	Retired IA-64 Instructions	7-45
IA32_INST_RETIRED	Retired IA-32 Instructions	7-45
ISA_TRANSITIONS	IA-64 to IA-32 ISA Transitions	7-48

Table 7-2 defines IPC and average instructions/cycles per ISA transition metrics.

Table 7-2. IA-64 and IA-32 Instruction Set Execution and Retirement Performance Metrics

Performance Metric	Performance Monitor Equation
IA-64 Instruction per Cycle	IA64_INST_RETIRED / CPU_CYCLES[IA-64 only]
IA-32 Instruction per Cycle	IA32_INST_RETIRED / CPU_CYCLES[IA-32 only]
Average IA-64 Instructions/Transition	IA64_INST_RETIRED/ (ISA_TRANSITIONS*2)
Average IA-32 Instructions/Transition	IA32_INST_RETIRED/ (ISA_TRANSITIONS*2)
Average IA-64 Cycles/Transition	CPU_CYCLES[IA64]/ (ISA_TRANSITIONS*2)
Average IA-32 Cycles/Transition	CPU_CYCLES[IA32]/ (ISA_TRANSITIONS*2)

7.3 Instruction Execution

This section describes events related to instruction issue and retirement (Table 7-3, Table 7-4) multi-media and FP (Table 7-5), data and control speculation (Table 7-7), as well as memory monitors (Table 7-9).

Table 7-3. Instruction Issue and Retirement Events

Decode, Issue, Retirement Monitors	Description	Page
INST_DISPERSED	Instructions Dispersed	7-47
EXPL_STOPS_DISPERSED	Explicit Stops Dispersed	7-44
ALL_STOPS_DISPERSED	Implicit and Explicit Stops Dispersed	7-19
IA64_TAGGED_INST_RETIRED	Retired Tagged IA-64 Instructions	7-46
NOPS_RETIRED	Retired Nop Instructions	7-59
PREDICATE_SQUASHED_RETIRED	Instructions Squashed Due to Predicate Off	7-60
RSE_REFERENCES_RETIRED	RSE Accesses	7-61
RSE_LOADS_RETIRED	RSE Load Accesses	7-60

Table 7-4. Instruction Issue and Retirement Events (Derived)

Decode, Issue, Retirement Monitors	Description	Performance Monitor Equation
RSE_STORES_RETIRED.d	RSE Store Accesses	RSE_REFERENCES_RETIRED - RSE_LOADS_RETIRED

Instruction cache lines are delivered to the execution core and are dispersed to the Itanium processor functional units. The number of dispersed instructions (INST_DISPERSED) on every cycle depends on the stops in the instruction stream (EXPL_STOPS_DISPERSED) as well as functional unit availability. Resource limitations and branch bundles (regardless of prediction) force a break in the instruction dispersal. Therefore, they are known as implicit stops, and can be computed as ALL_STOPS_DISPERSED - EXPL_STOPS_DISPERSED.

Retired instruction counts (IA64_TAGGED_INST_RETIRED, NOPS_RETIRED) are based on tag information specified by the address range check and opcode match facilities. The tagged retired instruction counts include predicated off instructions. A separate event (PREDICATE_SQUASHED_RETIRED) is provided to count predicated off instructions. RSE_REFERENCES_RETIRED counts the number of retired RSE operations.

There are two ways to count the total number of retired IA-64 instructions. Either the untagged IA64_INST_RETIRED event can be used or the IA64_TAGGED_INST_RETIRED event can be used by setting up the PMC_8 opcode match register to its don't care setting.

The FP monitors listed in Table 7-5 (FP_SIR_FLUSH, FP_FLUSH_TO_ZERO) capture dynamic information about pipeline flushes and flush-to-zero occurrences due to floating-point operations. FP_OPS_RETIRED.d is a derived event that counts the number of retired FP operations.

Table 7-5. Floating-point Execution Monitors

Floating-Point Monitors	Description	Page
FP_FLUSH_TO_ZERO	FP Result Flushed to Zero	7-44
FP_SIR_FLUSH	FP SIR Flushes	7-45

Table 7-6. Floating-point Execution Monitors (Derived)

Floating-Point Monitors	Description	Performance Monitor Equation
FP_OPS_RETIRED.d	FP Operations Retired	(4 * FP_OPS_RETIRED_HI) + FP_OPS_RETIRED_LO

As Table 7-7 describes, monitors for control and data speculation capture dynamic run-time information: the number of failed chk.s instructions (INST_FAILED_CHKS_RETIRED.ALL), the number of advanced load checks and check loads (ALAT_INST_CHKA_LDC.ALL) and failed advanced load checks and check loads (ALAT_INST_FAILED_CHKA_LDC.ALL) as seen by the ALAT. The number of retired chk.s instructions is monitored by the IA64_TAGGED_INST_RETIRED event with the appropriate opcode mask. Since the Itanium processor ALAT is updated by operations on mispredicted branch paths the number of advanced load checks and check loads needs an explicit event (ALAT_INST_CHKA_LDC.ALL). Finally, the ALAT_REPLACEMENT.ALL event can be used to monitor ALAT overflows.

Table 7-7. Control and Data Speculation Monitors

Control and Data Speculation Monitors	Description	Page
INST_FAILED_CHKS_RETIRED.ALL	Failed Speculative Check Loads	7-47
ALAT_INST_CHKA_LDC.ALL	Advanced Load Checks and Check Loads	7-18
ALAT_INST_FAILED_CHKA_LDC.ALL	Failed Advanced Load Checks and Check Loads	7-19
ALAT_REPLACEMENT.ALL	ALAT Entries Replaced by Any Instruction	7-17

Using an instruction type unit mask the four control and data speculation events can be constrained to monitor integer, FP or all speculative instructions. With the Itanium processor speculation monitors, the performance metrics described in Table 7-8 can be computed.

Table 7-8. Control/Data Speculation Performance Metrics

Performance Metric	Performance Monitor Equation
Control Speculation Miss Ratio	INST_FAILED_CHKS_RETIRED.ALL / (IA64_TAGGED_INST_RETIRED[chk.s]-PREDICATE_SQUASHED_RETIRED[chk.s])
Data Speculation Miss Ratio	ALAT_INST_FAILED_CHKA_LDC.ALL / ALAT_INST_CHKA_LDC.ALL
ALAT Capacity Miss Ratio	ALAT_REPLACEMENT.ALL/ IA64_TAGGED_INST_RETIRED[ld.a,ld.sa,ld.c.nc, ldf.a, ldf.sa, ldf.c.nc]-PREDICATE_SQUASHED_RETIRED[ld.a, ld.sa, ld.c.nc, ldf.a, ldf.sa, ldf.c.nc])

The equations described in Table 7-8 for Control Speculation Miss Ratio and ALAT Capacity Miss Ratio involve subtracting PREDICATE_SQUASHED_RETIRED[some inst] from IA64_TAGGED_INST_RETIRED[some inst]. This is done because IA64_TAGGED_INST_RETIRED includes predicated off instructions in its count, which do not update architectural state and hence need to be discounted in computing any performance metric. Using the opcode matcher in PMC8 with PREDICATE_SQUASHED_RETIRED (along with IA64_TAGGED_INST_RETIRED) allows us to count the number of predicated off instances of that instruction as well. Note that computing the ALAT Capacity Miss Ratio will require multiple runs in order to obtain all the terms in the equation. This is done to the limitations imposed by the opcode matcher.

Finally, Table 7-9 defines six memory instruction retirement events to count retired loads and stores. These counts include RSE operations. The load counts include failed check load instructions.

Table 7-9. Memory Events

Memory Monitors	Description	Page
LOADS_RETIRED	Retired Loads	7-58
STORES_RETIRED	Retired Stores	7-61
UC_LOADS_RETIRED	Retired Uncacheable Loads	7-61
UC_STORES_RETIRED	Retired Uncacheable Stores	7-61
MISALIGNED_LOADS_RETIRED	Retired Unaligned Load Instructions	7-58
MISALIGNED_STORES_RETIRED	Retired Unaligned Store Instructions	7-58

7.4 Cycle Accounting Events

As described in Section 6.1.1.4, “Cycle Accounting”, the Itanium processor provides eight directly measured stall cycle monitors. Table 7-10 lists the cycle accounting events.

The Itanium processor classifies every clock cycle into one of 4 cycle counters, namely DEPENDENCY_ALL_CYCLE, MEMORY_CYCLE, UNSTALLED_BACKEND_CYCLE, and PIPELINE_ALL_FLUSH_CYCLE. The values of these 4 counters should add up to CPU_CYCLES.

DEPENDENCY_ALL_CYCLE counts the number of cycles lost to instruction dispersal breaks (including both explicit and implicit stops), FP-related flushes and scoreboard stalls on GR or FR dependencies on non-load instructions. That is, the monitor does not count stalls that occur when an instruction is waiting for source operands from the memory subsystem. Also note that this monitor

does not count the number of cycles when the machine is executing instructions without stalls or flushes. The DEPENDENCY_SCOREBOARD_CYCLE monitor operates similarly, but does not include instruction dispersal breaks.

Table 7-10. Stall Cycle Monitors

Stall Accounting Monitors	Description	Page
PIPELINE_BACKEND_FLUSH_CYCLE	Combination of Pipeline Flush Cycles caused by either a Branch Misprediction or an Exception	7-59
DATA_ACCESS_CYCLE	Data Access Stall Cycles	7-42
DEPENDENCY_SCOREBOARD_CYCLE	Scoreboard Dependency Cycles	7-43
INST_ACCESS_CYCLE	Instruction Access Cycles	7-46
PIPELINE_ALL_FLUSH_CYCLE	Combination of Pipeline Flush Cycles caused by either a front-end or a back-end source	7-59
MEMORY_CYCLE	Combined Memory Stall Cycles	7-58
DEPENDENCY_ALL_CYCLE	Scoreboard Dependency and Dispersal Break Cycles	7-42
UNSTALLED_BACKEND_CYCLE	Unstalled Back-end Cycles	7-62

MEMORY_CYCLE counts the number of cycles that the pipeline is stalled when instructions are waiting for source operands from the memory subsystem, and for pipeline flushes related to memory-access (L1D way mispredictions, DTC flushes). It also counts the number of clocks that the pipeline stalls for the Register Stack Engine to spill or fill registers to/from memory. The DATA_ACCESS_CYCLE monitor operates similarly, but excludes RSE activity.

UNSTALLED_BACKEND_CYCLE counts the number of cycles that the back-end is processing instructions without delay and the decoupling buffer between the front-end and back-end is empty. In this situation, any effect on the front-end will appear at the back-end of the pipeline. Thus, this monitor reflects the number of cycles where there are no back-end stalls or flushes, and the decoupling buffer is empty, regardless of whether the L1I and ITLB are being hit or missed. The INST_ACCESS_CYCLE monitor includes those cycles where there are no back-end stalls or flushes, the decoupling buffer is empty, and the front-end is stalled waiting on an L1I or ITLB miss.

PIPELINE_ALL_FLUSH_CYCLE counts the number of cycles lost to branch related resteers. Resteers can be classified as branch prediction resteers (which occur when the front-end correctly predicts a taken branch) or as branch misprediction resteers (which occur when the back-end determines that the front-end incorrectly predicted a taken or not-taken branch). Note that taken branches incorrectly predicted by the front-end will not be counted twice. The branch misprediction flush that occurs in the back-end will override the front-end bubble. The monitor also counts ALAT flushes, serialization flushes, MMU-IEU bypass flushes, failed control speculation flushes and other exception flushes. The monitor PIPELINE_BACKEND_FLUSH_CYCLE operates similarly, but excludes front-end resteers to correctly predicted branches (commonly known as “branch bubbles”).

Table 7-11 defines derived stall cycle accounting monitors in terms of directly measured monitors.

Table 7-11. Stall Cycle Monitors (Derived)

Stall Cycle Monitors (Derived)	Description	Performance Monitor Equation
RSE_ACTIVE_CYCLE.d	RSE Active Cycles	MEMORY_CYCLE - DATA_ACCESS_CYCLE
ISSUE_LIMIT_CYCLE.d	Issue Limit Cycles	DEPENDENCY_ALL_CYCLE - DEPENDENCY_SCOREBOARD_CYCLE

Table 7-11. Stall Cycle Monitors (Derived) (Continued)

Stall Cycle Monitors (Derived)	Description	Performance Monitor Equation
TAKEN_BRANCH_CYCLE.d	Taken Branch Cycles	PIPELINE_ALL_FLUSH_CYCLE - PIPELINE_BACKEND_FLUSH_CYCLE
UNSTALLED_PIPELINE_CYCLE.d	Unstalled Pipeline Cycles	UNSTALLED_BACKEND_CYCLE - INST_ACCESS_CYCLE

7.5 Branch Events

The five measured Itanium processor branch events listed in Table 7-12 expand into over fifty measurable branch metrics by using the unit masks described on the event pages. BRANCH_PATH provides insight into the accuracy of taken/not-taken predicate predictions; unit masks allow classification by prediction, outcome and predictor type. BRANCH_PREDICTOR classifies how branches are predicted by different predictors as they move down the branch prediction pipeline; unit masks provide finer resolution and break down events into correct predictions, incorrect predicate predictions, and incorrect target predictions. BRANCH_MULTIWAY collects events exclusively for predictions on multiway branch bundles, from which their single-way counterparts can be derived. BRANCH_TAKEN_SLOT gives information regarding the position within a bundle that actually-taken branches occupy. BRANCH_EVENT counts the number of events captured in the branch trace buffer.

Table 7-13 defines derived branch monitors in terms of directly measure monitors.

Table 7-12. Branch Monitors

Branch Events	Description
BRANCH_PATH	Accuracy of predicate (taken/not-taken) predictions
BRANCH_PREDICTOR	Classification of how the branches are predicted in the pipeline
BRANCH_MULTIWAY	Details on multiway branch bundle predictions (details on single-way branch bundle predictions can be derived from this event)
BRANCH_TAKEN_SLOT	Location of taken branches (if any) in a bundle
BRANCH_EVENT	Branch Event Captured

Table 7-13. Branch Monitors (Derived)

Branch Events	Description	Performance Monitor Equation
BRANCH_MISPREDICTIONS.d	Branch Bundles Mispredicted	(BRANCH_PREDICTOR.ALL.ALL_PREDICTIONS - BRANCH_PREDICTOR.ALL.CORRECT_PREDICTIONS) or (BRANCH_PREDICTOR.ALL.WRONG_PATH + BRANCH_PREDICTOR.ALL.WRONG_TARGET)
BRANCH_1ST_STAGE_PREDICTIONS.d	Branch Bundles (Correctly or Incorrectly) Predicted in the 1st Pipeline Stage	BRANCH_PREDICTOR.1ST_STAGE.ALL_PREDICTIONS
BRANCH_1ST_STAGE_MISPREDICTIONS.d	Branch Bundles Incorrectly Predicted in the 1st Pipeline Stage	(BRANCH_PREDICTOR.ALL.ALL_PREDICTIONS - BRANCH_PREDICTOR.ALL.CORRECT_PREDICTIONS) or (BRANCH_PREDICTOR.1ST_STAGE.WRONG_PATH + BRANCH_PREDICTOR.1ST_STAGE.WRONG_TARGET)
BRANCH_2ND_STAGE_PREDICTIONS.d	Branch Bundles (Correctly or Incorrectly) Predicted in the 2nd Pipeline Stage	BRANCH_PREDICTOR.2ND_STAGE.ALL_PREDICTIONS

Table 7-13. Branch Monitors (Derived) (Continued)

Branch Events	Description	Performance Monitor Equation
BRANCH_2ND_STAGE_MISPREDICTIONS.d	Branch Bundles Incorrectly Predicted in the 2nd Pipeline Stage	(BRANCH_PREDICTOR.ALL.ALL_PREDICTIONS - BRANCH_PREDICTOR.ALL.CORRECT_PREDICTIONS) or (BRANCH_PREDICTOR.2ND_STAGE.WRONG_PATH + BRANCH_PREDICTOR.2ND_STAGE.WRONG_TARGET)
BRANCH_3RD_STAGE_PREDICTIONS.d	Branch Bundles (Correctly or Incorrectly) Predicted in the 3rd Pipeline Stage	BRANCH_PREDICTOR.3RD_STAGE.ALL_PREDICTIONS
BRANCH_3RD_STAGE_MISPREDICTIONS.d	Branch Bundles Incorrectly Predicted in the 3rd Pipeline Stage	(BRANCH_PREDICTOR.ALL.ALL_PREDICTIONS - BRANCH_PREDICTOR.ALL.CORRECT_PREDICTIONS) or (BRANCH_PREDICTOR.3RD_STAGE.WRONG_PATH + BRANCH_PREDICTOR.3RD_STAGE.WRONG_TARGET)
BRANCH_MULTIWAY_COMPONENT.d	Multiway Branch Bundle Predictions Relative to All Prediction	BRANCH_MULTIWAY.ALL_PATHS.ALL_PREDICTIONS / BRANCH_PREDICTOR.ALL.ALL_PREDICTIONS

All branch events can be qualified by instruction address range and opcode matching as described in Section 6.1.3, “Event Qualification”. Since the instruction address range check is bundle granular, qualification of multiway branches by address range is straightforward. However, for opcode matching purposes, multiway branches (MBB or BBB bundle templates) are qualified up to and including the first taken branch as follows:

```
   ((address range and opcode match on instruction slot 0)
      and (branch in slot 0 is taken))
or ((address range and opcode match on instruction slot 1)
      and (branch in slot 0 is NOT taken)
      and (branch in slot 1 is taken))
or  ((address range and opcode match on instruction slot 0 or 1 or 2)
      and (branch in slot 0 is NOT taken)
      and (branch in slot 1 is NOT taken))
```

7.6 Memory Hierarchy

This section summarizes events related to the Itanium processor’s memory hierarchy. The memory hierarchy events are grouped as follows:

- L1 Instruction Cache and Prefetch (Section 7.6.1)
- L1 Data Cache (Section 7.6.2)
- L2 Unified Cache (Section 7.6.3)
- L3 Unified Cache (Section 7.6.4)

An overview of the Itanium processor’s three-level memory hierarchy and its event monitors is shown in Figure 7-1. The instruction and the data stream work through separate L1 caches. The L1 data cache is a write-through cache. A unified L2 cache serves both the L1 instruction and data caches, and is backed by a large unified L3 cache. Events for individual levels of the cache hierarchy are described in the following three sections. They can be used to compute the most common cache performance ratios summarized in Table 7-15.

For common performance metrics not directly measured by hardware, the equations listed in Table 7-14 can be used.

Figure 7-1. Event Monitors in the Itanium™ Processor Memory Hierarchy

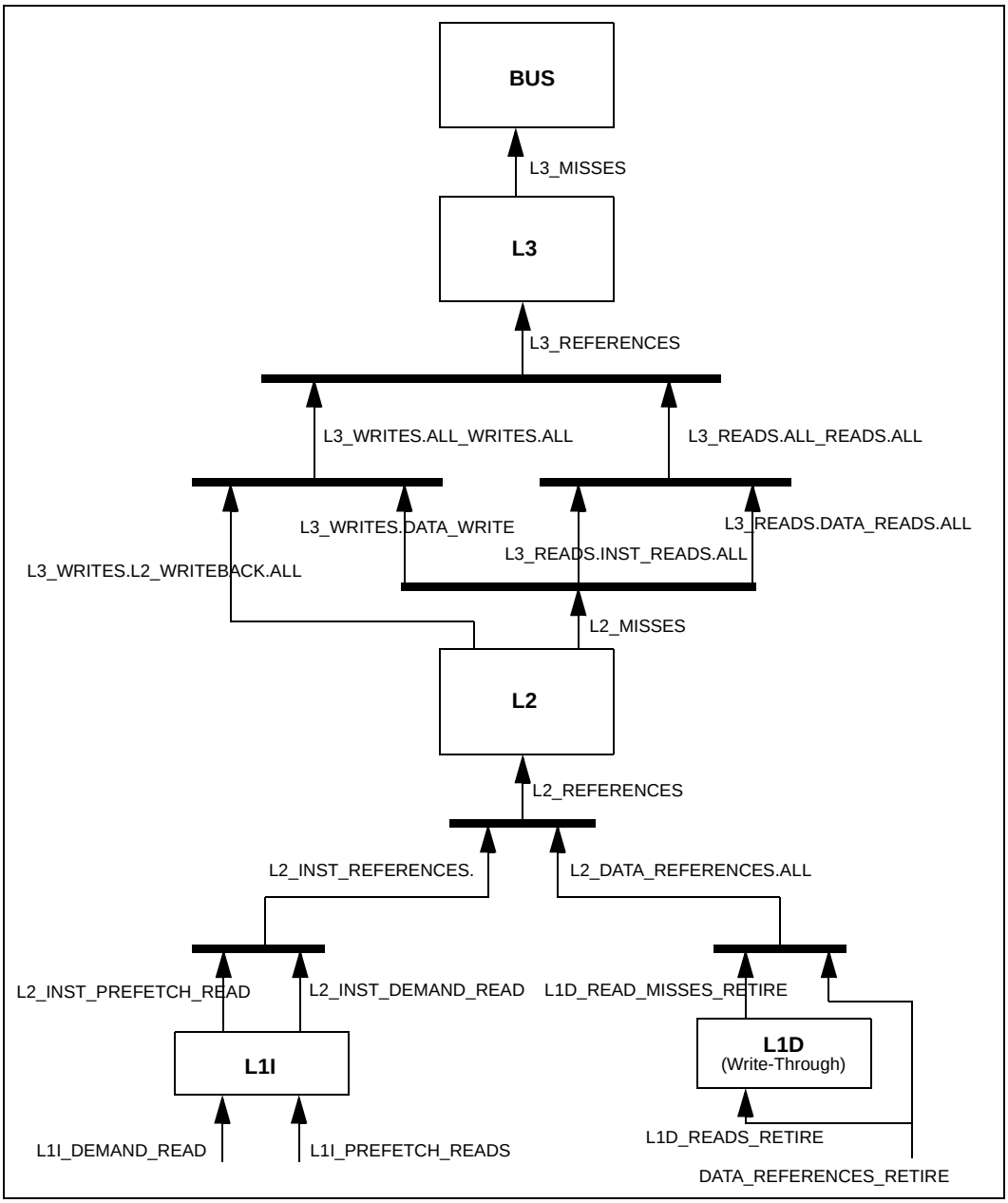


Table 7-14. Derived Memory Hierarchy Monitors

Memory Hierarchy Monitors (Derived)	Description	Performance Monitor Equation
L1I_REFERENCES.d	L1 Instruction Cache References	L1I_PREFETCH_READS + L1I_DEMAND_READS
L2_INST_REFERENCES.d	L2 Instruction References	L2_INST_DEMAND_READS + L2_INST_PREFETCH_READS
L3_DATA_REFERENCES.d	L3 Data References	L3_WRITES.DATA_WRITES.ALL+ L3_READS.DATA_READS.ALL
L3_CORRECTION_RATIO.d	L3 Correction Ratio	L2_MISSES/(L3_REFERENCES-L3_WRITES.L2_WRITEBACK.ALL)

Because Itanium performance monitors for the L2 cache contain secondary misses from L1, and performance monitors for L3 do not, one cannot directly compare L2 and L3 performance monitors. In the performance metrics that follow in Table 7-15 where possible only events from an L2 view or an L3 view are used in computing the metric. However, several useful metrics cannot be calculated accurately this way. For those metrics we compute a correction ratio to scale L3 events to approximate a value that contains secondary misses. Table 7-14 contains the definition of this correction ratio.

Table 7-15. Cache Performance Ratios

Performance Metric	Performance Monitor Equation
L1I Miss Ratio	L2_INST_REFERENCES.d/L1I_REFERENCES.d
L1I Demand Miss Ratio	L2_INST_DEMAND_READS / L1I_DEMAND_READS
L1I Prefetch Miss Ratio	L2_INST_PREFETCH_READS/ L1I_PREFETCH_READS
L1D Read Miss Ratio	L1D_READ_MISSES_RETIRED / L1D_READS_RETIRED
L2 Miss Ratio	L2_MISSES / L2_REFERENCES
Approximate L2 Data Miss Ratio	(L3_DATA_REFERENCES.d / L2_DATA_REFERENCES.ALL)*L3_CORRECTION_RATIO.d
Approximate L2 Instruction Miss Ratio (includes prefetches)	(L3_READS.INST_READS.ALL / L2_INST_REFERENCES.d)*L3_CORRECTION_RATIO.d
Approximate L2 Data Read Miss Ratio	(L3_READS.DATA_READS.ALL / L2_DATA_REFERENCES.READS) * L3_CORRECTION_RATIO.d
Approximate L2 Data Write Miss Ratio	(L3_WRITES.DATA_WRITES.ALL / L2_DATA_REFERENCES.WRITES)*L3_CORRECTION_RATIO.d
L2 Instruction Ratio	L2_INST_REFERENCES.d/ L2_REFERENCES
L2 Data Ratio	L2_DATA_REFERENCES.ALL / L2_REFERENCES
L3 Miss Ratio	L3_MISSES /(L3_REFERENCES-L3_WRITES.L2_WRITEBACK.ALL)
L3 Data Miss Ratio	(L3_READS.DATA_READS.MISS + L3_WRITES.DATA_WRITES.MISS) / L3_DATA_REFERENCES.d
L3 Instruction Miss Ratio	L3_READS.INST_READS.MISS / L3_READS.INST_READS.ALL
L3 Data Read Ratio	L3_READS.DATA_READS.ALL / L3_DATA_REFERENCES.d
L3 Data Ratio	L3_DATA_REFERENCES.d / L3_REFERENCES
L3 Instruction Ratio	L3_READS.INST_READS.ALL / L3_REFERENCES

7.6.1 L1 Instruction Cache and Prefetch

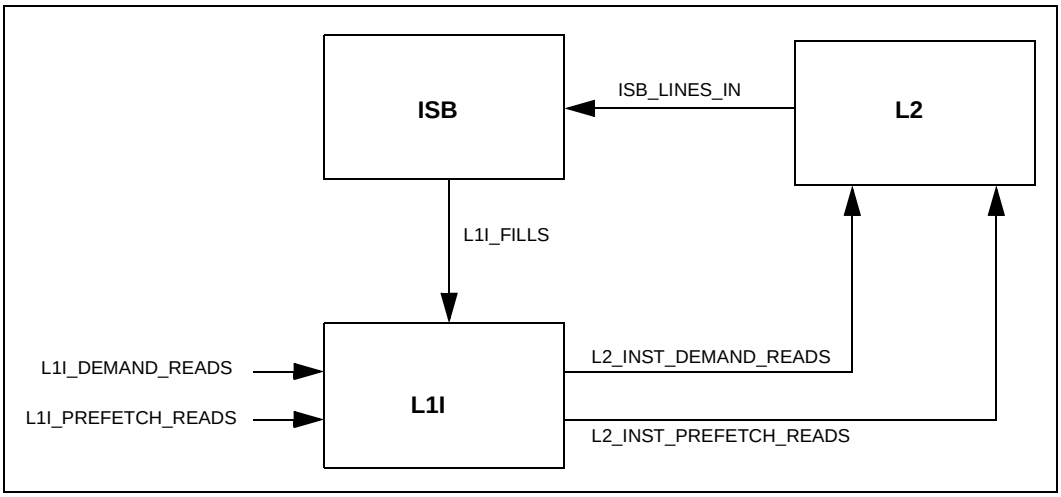
Table 7-16 and Figure 7-2 describes and summarizes the events that the Itanium processor provides to monitor the L1 instruction cache demand fetch and prefetch activity. Table 7-14 lists pertinent derived events. The instruction fetch monitors distinguish between demand fetch (L1I_DEMAND_READS, L2_INST_DEMAND_READS) and prefetch activity (L1I_PREFETCH_READS, L2_INST_PREFETCH_READS). The amount of data returned from the L2 into the L1 instruction cache and the Instruction Streaming Buffer is monitored by two events (L1I_FILLS, ISB_LINES_IN). The INSTRUCTION_EAR_EVENTS monitor (not shown in Figure 7-2) counts how many instruction cache or ITLB misses are captured by the instruction event address register.

The L1 instruction cache and prefetch events can be qualified by the instruction address range check, but not by the opcode matcher. Since instruction cache and prefetch events occur early in the processor pipeline, they include events caused by speculative, wrong-path as well as predicated off instructions. Since the address range check is not based on actually retired, but speculative instruction addresses, event counts may be inaccurate when the range checker is confined to address ranges smaller than the length of the processor pipeline (see Section 6.2.4, “IA-64 Instruction Address Range Check Register (PMC[13])” for details).

Table 7-16. L1 Instruction Cache and Instruction Prefetch Monitors

L1I and I-Prefetch Monitors	Description	Page
L1I_DEMAND_READS	L1I and ISB Instruction Demand Lookups	7-50
L1I_FILLS	L1 Instruction Cache Fills	7-50
L2_INST_DEMAND_READS	L2 Instruction Demand Fetch Requests	7-52
INSTRUCTION_EAR_EVENTS	Instruction EAR Events	7-48
L1I_PREFETCH_READS	L1I and ISB Instruction Prefetch Lookups	7-50
L2_INST_PREFETCH_READS	L2 Instruction Prefetch Requests	7-52
ISB_LINES_IN	Instruction Streaming Buffer Lines In	7-48

Figure 7-2. L1 Instruction Cache and Prefetch Monitors



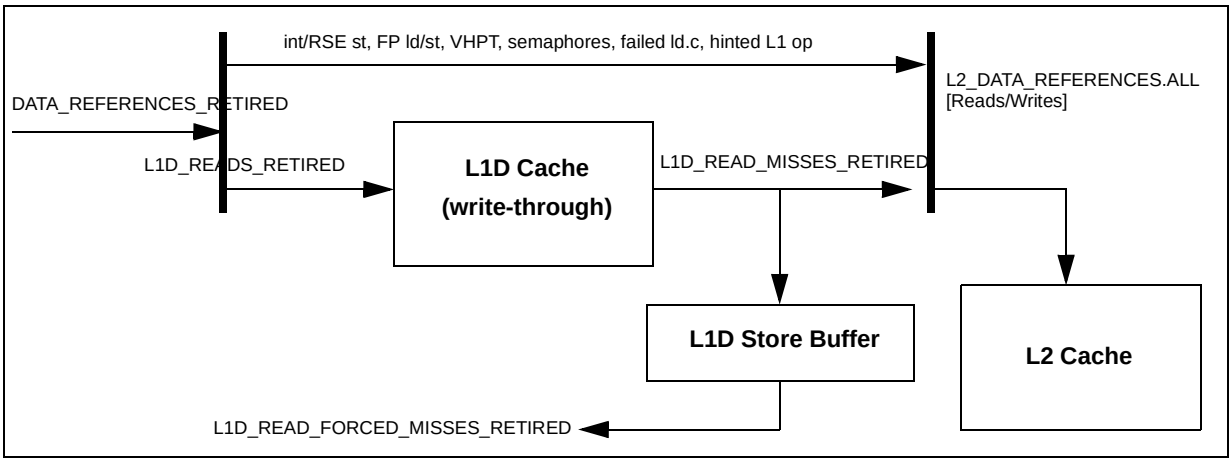
7.6.2 L1 Data Cache

Table 7-17 lists the Itanium processor's seven L1 data cache monitors. As shown in Figure 7-3, the write-through L1 data cache services cacheable loads, Integer and RSE stores, FP memory operations, VHPT references, semaphores, check loads and hinted L2 memory references. DATA_REFERENCES_RETIRED is the number of issued data memory references. L1 data cache reads (L1D_READS_RETIRED) and L1 data cache misses (L1D_READ_MISSES_RETIRED) monitor the read hit/miss rate for the L1 data cache. The number of L2 data references (L2_DATA_REFERENCES.ALL) is the number of data requests prior to cache line merging. Unit mask selections allow breaking down into reads and writes. The DATA_EAR_EVENTS monitor (not shown in Figure 7-3) counts how many data cache or DTLB misses are captured by the data event address register. RSE operations are included in all data cache monitors, but are not broken down explicitly.

Table 7-17. L1 Data Cache Monitors

L1D Monitors	Description	Page
DATA_REFERENCES_RETIRED	Retired Data Memory References	7-42
L1D_READS_RETIRED	L1 Data Cache Reads	7-49
L1D_READ_MISSES_RETIRED	L1 Data Cache Read Misses	7-49
PIPELINE_FLUSH.L1D_WAY_MISPREDICT	Pipeline Flush	7-60
L1D_READ_FORCED_MISSES_RETIRED	L1 Data Cache Forced Load Misses	7-49
L2_DATA_REFERENCES.ALL	L2 Data Read and Write References	7-50
DATA_EAR_EVENTS	L1 Data Cache EAR Events	7-42

Figure 7-3. L1 Data Cache Monitors



7.6.3 L2 Unified Cache

Table 7-18 summarizes the directly-measured events that monitor the Itanium processor L2 cache. Table 7-14 lists pertinent derived events. Refer to Figure 7-1 for a graphical view of the L2 cache monitors.

L2_REFERENCES, L2_INST_DEMAND_READS, L2_INST_PREFETCH_READS, L2_DATA_REFERENCES.ALL, and L2_MISSES are all counted in terms of number of requests

seen by the L2. L2_FLUSHES and L2_FLUSH_DETAILS count and break-down the number of L2 flushes due to address conflicts, store buffer conflicts, bus rejects, and other reasons. L1D_READ_FORCED_MISSES_RETIRED counts the number of loads that were bypassed from an earlier store.

Table 7-18. L2 Cache Monitors

L1 Monitors	Description	Page
L2_REFERENCES	L2 References	7-53
L2_INST_PREFETCH_READS	L2 Instruction Prefetch Requests	7-52
L2_INST_DEMAND_READS	L2 Instruction Demand Fetch Requests	7-52
L2_DATA_REFERENCES.ALL	L2 Data Read and Write References	7-50
L2_DATA_REFERENCES.READS	L2 Data Read References	7-51
L2_DATA_REFERENCES.WRITES	L2 Data Write References	7-51
L2_MISSES	L2 Misses	7-52
L2_FLUSHES	L2 Flushes	7-52
L2_FLUSH_DETAILS	L2 Flush Details	7-51

7.6.4 L3 Unified Cache

Table 7-19 summarizes the directly-measured L3 cache events. Table 7-14 lists pertinent derived events. Refer to Figure 7-1 for a graphical view of the L3 cache monitors.

Table 7-19. L3 Cache Monitors

L2 Monitors	Description	Page
L3_REFERENCES	L3 References	7-55
L3_MISSES	L3 Misses	7-53
L3_LINES_REPLACED	L3 Cache Lines Replaced	7-53
L3_READS.ALL_READS.ALL	Instruction and Data L3 Reads	7-53
L3_READS.ALL_READS.HIT	Instruction and Data L3 Read Hits	7-54
L3_READS.ALL_READS.MISS	Instruction and Data L3 Read Misses	7-54
L3_READS.DATA_READS.ALL	Data L3 Reads	7-54
L3_READS.DATA_READS.HIT	Data L3 Read Hits	7-54
L3_READS.DATA_READS.MISS	Data L3 Read Misses	7-54
L3_READS.INST_READS.ALL	Instruction L3 Reads	7-55
L3_READS.INST_READS.HIT	Instruction L3 Read Hits	7-55
L3_READS.INST_READS.MISS	Instruction L3 Read Misses	7-55
L3_WRITES.ALL_WRITES.ALL	L3 Writes	7-56
L3_WRITES.ALL_WRITES.HIT	L3 Write Hits	7-56
L3_WRITES.ALL_WRITES.MISS	L3 Write Misses	7-56
L3_WRITES.L2_WRITEBACK.ALL	L3 Writebacks	7-56
L3_WRITES.L2_WRITEBACK.HIT	L3 Writeback Hits	7-57
L3_WRITES.L2_WRITEBACK.MISS	L3 Writeback Misses	7-57
L3_WRITES.DATA_WRITES.ALL	L3 Data Writes	7-57
L3_WRITES.DATA_WRITES.HIT	L3 Data Write Hits	7-57
L3_WRITES.DATA_WRITES.MISS	L3 Data Write Misses	7-57

7.6.5 Frontside Bus

Table 7-20 lists the frontside bus or system bus transaction monitors.

Table 7-20. Bus Events

L2 Monitors	Description	Page
BUS_ALL	Bus Transactions	7-33
BUS_PARTIAL	Bus Partial Transactions	7-37
BUS_BURST	Bus Burst Transactions	7-35
BUS_MEMORY	Bus Memory Transactions	7-37
BUS_RD_ALL	Bus Read Transactions	7-37
BUS_RD_DATA	Bus Read Data Transactions	7-37
BUS_RD_PRTL	Bus Read Partial Transactions	7-39
BUS_RD_HIT	Bus Read Hit Clean Non-local Cache Transactions	7-38
BUS_RD_HITM	Bus Read Hit Modified Non-local Cache Transactions	7-38
BUS_RD_INVAL	Bus Read Invalidated Line	7-38
BUS_RD_INVAL_HITM	Bus BIL Transaction Results in HITM	7-39
BUS_RD_INVAL_BST	Bus BRIL Burst Transactions	7-38
BUS_RD_INVAL_BST_HITM	Bus BRIL Burst Transaction Results in HITM	7-39
BUS_HITM	Bus Hit Modified Line Transactions	7-35
BUS_WR_WB	Bus Write Back Transactions	7-41
BUS_SNOOPS_HITM	Bus Snoops Hit Modified Cache Line	7-40
BUS_SNOOPS	Bus Snoops Total	7-40
BUS_SNOOP_STALL_CYCLES	Bus Snoop Stall Cycles	7-40
BUS_SNOOPQ_REQ	Bus Snoop Queue Requests, Category: Frontside Bus	7-41
BUS_BRQ_READ_REQ_INSERTED	BRQ Requests Inserted	7-34
BUS_IO	IA-32 Compatible I/O Bus Transactions	7-35
BUS_RD_IO	IA-32 Compatible I/O Read Transactions	7-39
BUS_LOCK	IA-32 Compatible Bus Lock Transactions	7-36
BUS_LOCK_CYCLES	IA-32 Compatible Bus Lock Cycles	7-36

Table 7-21 lists the derived frontside bus transaction monitors.

Table 7-21. Frontside Bus Monitors (Derived)

Bus Monitors (Derived)	Description	Performance Monitor Equation
BUS_RD_INSTRUCTIONS.d	Bus Read Instructions	BUS_RD_ALL - BUS_RD_DATA
BUS_RD_INVAL_MEMORY.d	Bus BIL Transaction Satisfied from Memory	BUS_RD_INVAL - BUS_RD_INVAL_HITM
BUS_RD_INVAL_BST_MEMORY.d	Bus BRIL Burst Transaction Satisfied from Memory	BUS_RD_INVAL_BST - BUS_RD_INVAL_BST_HITM
BUS_ADDR_BPRI.d	Bus Used by I/O Agent	BUS_MEMORY.IOAGENT
BUS_IOQ_LIVE_REQ.d	In-order Bus Queue Requests	BUS_IOQ_LIVE_REQ_HI * 4 + BUS_IOQ_LIVE_REQ_LO
BUS_BRQ_LIVE_REQ.d	BRQ Live Requests	BUS_BRQ_LIVE_REQ_HI * 4 + BUS_BRQ_LIVE_REQ_LO

Most of the bus events in Section 7.6.5 can be qualified by the bus transaction initiator using the three way unit mask as described in Table 7-22.

Table 7-22. Unit Masks for Qualifying Bus Transaction Events by Initiator

Selection	PMC.umask[19:16]	Description
ANY	x001	Counts all bus transactions (initiated by any processor or non-processor bus masters)
SELF	x010	Counts bus transactions initiated by the local processor only
IO	x100	Counts bus transactions from IO agents, i.e. non-processor bus masters

Table 7-23 defines the conventions that will be used when describing the Itanium procesor frontside bus transaction monitors in Section 7.6.5.

Table 7-23. Conventions for Frontside Bus Transactions

Name	Description
BRL	Memory Read (64 byte bursts). Includes code fetches and data loads from WB memory.
BRIL	Memory Read & Invalidate (64 byte bursts). Also known as read for ownership (RFO).
BIL	Memory Read & Invalidate (0 byte sized transaction). Caused by flush cache (`fc`) instruction only.
BWL	Memory Write (64 byte bursts). Explicit writebacks/coalesced writes.
BRP	Partial Memory Reads (<64 byte transactions). Typically, uncacheable reads.
BWP	Partial Memory Write (<64 byte transactions). Typically, uncacheable writes.
IORD	Partial IO Read (<64 byte transactions). Uncacheable read to IO port space.
IOWR	Partial IO Write (<64 byte transactions). Uncacheable write to IO port space.

Other transactions besides those listed in Table 7-23 include Deferred Reply, Special Transactions, Interrupt, Interrupt Acknowledge, and Purge TC. For the bus performance monitors in Section 7.6.5, note that the monitors will count if any transaction gets a retry response from the priority agent.

To support the analysis of snoop traffic in a multiprocessor system, the Itanium processor provides local processor and remote response monitors. The local processor snoop events (BUS_SNOOPS_HITM, BUS_SNOOPS, BUS_SNOOPQ_REQ) monitor inbound snoop traffic. The remote response events (BUS_RD_HIT, BUS_RD_HITM, BUS_RD_INVAL_HITM, BUS_RD_INVAL_BST_HITM) monitor the snoop responses of other processors to bus transactions that the monitoring processor originated. Table 7-24 summarizes the remote snoop events by bus transaction.

Table 7-24. Bus Events by Snoop Response

Remote Processor Response	BRL	BIL	BRIL
HIT	BUS_RD_HIT	NA	NA
HITM	BUS_RD_HITM	BUS_RD_INVAL_HITM	BUS_RD_INVAL_BST_HITM
ALL	BUS_RD_ALL	BUS_RD_INVAL	BUS_RD_INVAL

With the Itanium processor frontside bus monitors, the performance metrics described in Table 7-25 can be computed.

Table 7-25. Bus Performance Metrics

Performance Metric	Performance Monitor Equation
Cacheable Data Fetch Bus Transaction Ratio	BUS_RD_DATA/BUS_ALL or BUS_RD_DATA/BUS_MEMORY
Partial Access Ratio	BUS_PARTIAL/BUS_MEMORY
Read Partial Access Ratio	BUS_RD_PRTL/BUS_MEMORY
Read Hit To Shared Line Ratio	BUS_RD_HIT/BUS_RD_ALL or BUS_MEMORY
Read Hit to Modified Line Ratio	BUS_RD_HITM/BUS_RD_ALL or BUS_RD_HITM/BUS_MEMORY
BIL Ratio	BUS_RD_HIT/BUS_MEMORY
BIL Hit to Modified Line Ratio	BUS_RD_INVAL_HITM/BUS_MEMORY or BUS_RD_INVAL_HITM/BUS_RD_INVAL
BRIL Hit to Modified Line Ratio	BUS_RD_INVAL_BST_HITM/BUS_MEMORY or BUS_RD_INVAL_BST_HITM/BUS_RD_INVAL
Bus Modified Line Hit Ratio	BUS_RD_HITM/BUS_MEMORY or BUS_RD_HITM/BUS_BURST
Writeback Ratio	BUS_WR_WB/BUS_MEMORY or BUS_WR_WB/BUS_BURST
Cacheable Read Ratio	(BUS_RD_ALL + BUS_RD_INVAL_BST)/BUS_MEMORY
I/O Cycle Ratio	BUS_IO/BUS_ALL
I/O Read	BUS_RD_IO/BUS_ALL

7.7 System Events

Table 7-26 lists the directly measurable system and TLB events. Table 7-27 lists pertinent derived events. The debug register match events count how often the address in any instruction or data break-point register (IBR or DBR) matches the current retired instruction pointer (CODE_DEBUG_REGISTER_MATCHES.d) or the current data memory address (DATA_DEBUG_REGISTER_MATCHES.d). PIPELINE_FLUSH counts the number of times the Itanium processor pipeline is flushed due to a data translation cache miss, L1 data cache way mispredict, an exception flush or an instruction serialization event. CPU_CPL_CHANGES counts the number of privilege level transitions due to interruptions, system calls (epc) and returns (demoting branch), and `rfi` instructions. CPU_CYCLES counts the number of cycles the CPU is not powered down or in light HALT state.

Table 7-26. System and TLB Monitors

System and Processor TLB Monitors	Description	Page
PIPELINE_FLUSH	Pipeline Flush	7-60
CPU_CPL_CHANGES	Privilege level changes	7-41
CPU_CYCLES	CPU Cycles	7-41
ITLB_MISSES_FETCH	ITLB Demand Misses	7-49
ITLB_INSERTS_HPW	Hardware Page Walker Inserts into the ITLB	7-48

Table 7-26. System and TLB Monitors (Continued)

System and Processor TLB Monitors	Description	Page
DTC_MISSES	DTC Misses	7-43
DTLB_MISSES	DTLB Misses	7-43
DTLB_INSERTS_HPW	Hardware Page Walker Inserts into the DTLB	7-43

Table 7-27 defines derived system and TLB events that are computed from events directly measured by hardware.

Table 7-27. System and TLB Monitors (Derived)

Derived Memory Hierarchy Monitors	Description	Performance Monitor Equation
CODE_DEBUG_REGISTER_MATCHES.d	Code Debug Register Matches	IA64_TAGGED_INST_RETIRED
DATA_DEBUG_REGISTER_MATCHES.d	Data Debug Register Matches	LOADS_RETIRED + STORES_RETIRED
ITLB_REFERENCES.d	ITLB References	L1I_DEMAND_READS
ITLB_EAR_EVENT.d	ITLB EAR Event	INSTRUCTION_EAR_EVENTS
DTLB_REFERENCES.d	DTLB References	DATA_REFERENCES_RETIRED
DTLB_EAR_EVENT.d	DTLB EAR Event	DATA_EAR_EVENTS

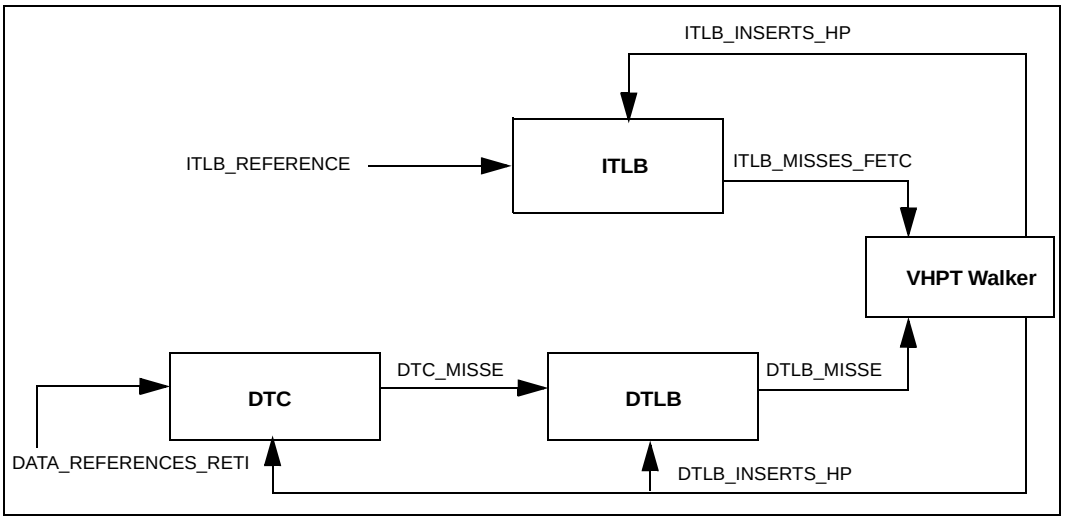
The Itanium processor instruction and data TLBs and the virtual hash page table walker are monitored by the events described in Table 7-26 and Table 7-27. Figure 7-4 gives a graphical summary. Table 7-28 lists the TLB performance metrics that can be computed using these events.

ITLB_REFERENCES.d and DTLB_REFERENCES.d are derived from the respective instruction/data cache access events. Note that ITLB_REFERENCES.d does not include prefetch requests made to the L1I cache (L1I_PREFETCH_READS). This is because prefetches are cancelled when they miss in the ITLB and thus do not trigger VHPT walks or software TLB miss handling. ITLB_MISSES_FETCH and DTLB_MISSES count TLB misses. ITLB_INSERTS_HPW and DTLB_INSERTS_HPW count the number of instruction/data TLB inserts performed by the virtual hash page table walker. The Itanium processor data TLB is a two level TLB; DTC_MISSES counts the number of first level data TLB misses.

Table 7-28. TLB Performance Metrics

Performance Metric	Performance Monitor Equation
ITLB Miss Ratio	ITLB_MISSES_FETCH / ITLB_REFERENCES.d
DTLB Miss Ratio	DTLB_MISSES / DTLB_REFERENCES.d
DTC Miss Ratio	DTC_MISSES / DTLB_REFERENCES.d

Figure 7-4. Instruction and Data TLB Monitors



7.8 Performance Monitor Event List

This section enumerates Itanium processor performance monitoring events.

ALAT_REPLACEMENT.ALL

- **Title:** ALAT Entries Replaced by Any Instruction, **Category:** Execution
- **Definition:** ALAT_REPLACEMENT.ALL counts the number of times an advanced load (`ld.a` or `ld.as` or `ldfp.a` or `ldfp.as`) or a no-clear check load (`ld.c.nc` and variants of `ldf.c.nc`) displaced a valid entry in the ALAT
- **Event Code:** 0x38, **Umask:** xx11, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: yes

ALAT_REPLACEMENT.FP

- **Title:** ALAT Entries Replaced by FP Instructions, **Category:** Execution
- **Definition:** ALAT_REPLACEMENT.FP counts the number of times a FP advanced load (`ldfp.a` or `ldfp.as`) or a no-clear FP check load (variants of `ldf.c.nc`) displaced a valid entry in the ALAT
- **Event Code:** 0x38, **Umask:** xx10, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: yes

ALAT_REPLACEMENT.INTEGER

- **Title:** ALAT Entries Replaced by Integer Instructions, **Category:** Execution
- **Definition:** ALAT_REPLACEMENT.INTEGER counts the number of times an integer advanced load (`ld.a` or `ld.as`) or a no-clear integer check load (`ld.c.nc`) displaced a valid entry in the ALAT
- **Event Code:** 0x38, **Umask:** xx01, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: yes

ALAT_INST_CHKA_LDC.ALL

- **Title:** Advanced Load Checks and Check Loads, **Category:** Execution
- **Definition:** ALAT_INST_CHKA_LDC.ALL counts the number of all advanced load checks (`chk.a`) and check loads in both clear and no-clear forms (`ld.c.clr` or `ld.c.nc`, including FP variants) as seen by the ALAT
- **Event Code:** 0x36, **Umask:** xx11, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no for `chk.a`, and yes for `ld.c`

ALAT_INST_CHKA_LDC.FP

- **Title:** FP Advanced Load Checks and Check Loads, **Category:** Execution
- **Definition:** ALAT_INST_CHKA_LDC.FP counts all FP advanced load checks (`chk.a`) and all FP check loads in both clear and no-clear forms (`ld.c.clr` or `ld.c.nc`, FP variants only) as seen by the ALAT
- **Event Code:** 0x36, **Umask:** xx10, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no for `chk.a`, and yes for `ld.c`

ALAT_INST_CHKA_LDC.INTEGER

- **Title:** Integer Advanced Load Checks and Check Loads, **Category:** Execution
- **Definition:** ALAT_INST_CHKA_LDC.INTEGER counts all integer advanced load checks (`chk.a`) and all integer check loads in both clear and no-clear forms (`ld.c.clr` or `ld.c.nc`, excluding FP variants) as seen by the ALAT
- **Event Code:** 0x36, **Umask:** xx01, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no for `chk.a`, and yes for `ld.c`

ALAT_INST_FAILED_CHKA_LDC.ALL

- **Title:** Failed Advanced Load Checks and Check Loads, **Category:** Execution
- **Definition:** ALAT_INST_FAILED_CHKA_LDC.ALL counts failed advanced load checks (`chk.a`) and failed check loads in both clear and no-clear forms (`ld.c.clr` or `ld.c.nc`, including FP variants) as seen by the ALAT
- **Event Code:** 0x37, **Umask:** xx11, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no for `chk.a`, and yes for `ld.c`

ALAT_INST_FAILED_CHKA_LDC.FP

- **Title:** Failed FP Advanced Load Checks and Check Loads, **Category:** Execution
- **Definition:** ALAT_INST_FAILED_CHKA_LDC.FP counts failed FP advanced load checks (`chk.a`) and failed FP check loads in both clear and no-clear forms (`ld.c.clr` or `ld.c.nc`, FP variants only) as seen by the ALAT
- **Event Code:** 0x37, **Umask:** xx10, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no for `chk.a`, and yes for `ld.c`

ALAT_INST_FAILED_CHKA_LDC.INTEGER

- **Title:** Failed Integer Advanced Load Checks and Check Loads, **Category:** Execution
- **Definition:** ALAT_INST_FAILED_CHKA_LDC.INTEGER counts the number of failed integer advanced load checks (`chk.a`) and failed integer check loads in both clear and no-clear forms (`ld.c.clr` or `ld.c.nc`, excluding FP variants) as seen by the ALAT
- **Event Code:** 0x37, **Umask:** xx01, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no for `chk.a`, and yes for `ld.c`

ALL_STOPS_DISPERSED

- **Title:** Implicit and Explicit Stops Dispersed, **Category:** Instruction Issue
- **Definition:** ALL_STOPS_DISPERSED counts the sum of explicit programmer-specified stops (EXPL_STOPS_DISPERSED) and dispersal breaks due to resource limitations and branch instructions (independent of their predicate prediction).The sum includes stops encountered during hardware speculative wrong-path execution (i.e., in the shadow of a flush)
- **Event Code:** 0x2F, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

BRANCH_EVENT

- **Title:** Branch Event Captured, **Category:** Branch
- **Definition:** BRANCH_EVENT counts the number of branch events, including multiway branches captured by the Branch Trace Buffer.
- **Event Code:** 0x11, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_MULTIWAY.ALL_PATHS.ALL_PREDICTIONS

- **Title:** All Branch Predictions on Multiway Bundles, **Category:** Branch
- **Definition:** BRANCH_MULTIWAY.ALL_PATHS.ALL_PREDICTIONS counts all branch predictions made on multiway branch bundles
- **Event Code:** 0x0E, **Umask:** 0000, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_MULTIWAY.ALL_PATHS.CORRECT_PREDICTIONS

- **Title:** Correct Branch Predictions on Multiway Bundles, **Category:** Branch
- **Definition:** BRANCH_MULTIWAY.ALL_PATHS.CORRECT_PREDICTIONS counts all branch predictions on multiway branch bundles that do not necessitate a back-end branch misprediction flush
- **Event Code:** 0x0E, **Umask:** 0001, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_MULTIWAY.ALL_PATHS.WRONG_PATH

- **Title:** Incorrect Predicate Predictions on Multiway Bundles, **Category:** Branch
- **Definition:** BRANCH_MULTIWAY.ALL_PATHS.WRONG_PATH counts the number of multiway branch bundles whose combined predicate is incorrectly predicted. This includes bundles where all branch instructions are predicted not-taken and any one instruction is actually taken, and those bundles where a branch instruction was predicted taken and either a prior branch instruction in the bundle was actually taken or the predicted instruction was not taken. In any event, the processor resteers the front-end to the correct target, i.e., a given multiway bundle can only be mispredicted once
- **Event Code:** 0x0E, **Umask:** 0010, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_MULTIWAY.ALL_PATHS.WRONG_TARGET

- **Title:** Incorrect Target Predictions on Multiway Bundles, **Category:** Branch
- **Definition:** BRANCH_MULTIWAY.ALL_PATHS.WRONG_TARGET counts the number of multiway branch bundles where a branch instruction is correctly predicted taken, but its target is incorrect
- **Event Code:** 0x0E, **Umask:** 0011, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_MULTIWAY.NOT_TAKEN.ALL_PREDICTIONS

- **Title:** All Branch Predictions on Not-Taken Multiway Bundles, **Category:** Branch
- **Definition:** BRANCH_MULTIWAY.NOT_TAKEN.ALL_PREDICTIONS is analogous to BRANCH_MULTIWAY.ALL_PATHS.ALL_PREDICTIONS, except it applies only to multiway branch bundles where all branch instructions are actually not taken
- **Event Code:** 0x0E, **Umask:** 1000, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_MULTIWAY.NOT_TAKEN.CORRECT_PREDICTIONS

- **Title:** Correct Branch Predictions on Not-Taken Multiway Bundles, **Category:** Branch
- **Definition:** BRANCH_MULTIWAY.NOT_TAKEN.CORRECT_PREDICTIONS is analogous to BRANCH_MULTIWAY.ALL_PATHS.CORRECT_PREDICTIONS, except it applies only to multiway branch bundles where all branch instructions are actually not taken
- **Event Code:** 0x0E, **Umask:** 1001, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_MULTIWAY.NOT_TAKEN.WRONG_PATH

- **Title:** Incorrect Predicate Predictions on Not-Taken Multiway Bundles, **Category:**

Branch

- **Definition:** BRANCH_MULTIWAY.NOT_TAKEN.WRONG_PATH is analogous to BRANCH_MULTIWAY.ALL_PATHS.WRONG_PATH, except it applies only to multiway branch bundles where all branch instructions are actually not taken
- **Event Code:** 0x0E, **Umask:** 1010, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_MULTIWAY.NOT_TAKEN.WRONG_TARGET

- **Title:** Incorrect Target Predictions on Not-Taken Multiway Bundles, **Category:** Branch
- **Definition:** BRANCH_MULTIWAY.NOT_TAKEN.WRONG_TARGET should always count zero, as not-taken branches do not specify a branch target
- **Event Code:** 0x0E, **Umask:** 1011, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_MULTIWAY.TAKEN.ALL_PREDICTIONS

- **Title:** All Branch Predictions on Taken Multiway Bundles, **Category:** Branch
- **Definition:** BRANCH_MULTIWAY.TAKEN.ALL_PREDICTIONS is analogous to BRANCH_MULTIWAY.ALL_PATHS.ALL_PREDICTIONS, except it applies only to multiway branch bundles where at least one branch instruction is taken
- **Event Code:** 0x0E, **Umask:** 1100, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_MULTIWAY.TAKEN.CORRECT_PREDICTIONS

- **Title:** Correct Branch Predictions on Taken Multiway Bundles, **Category:** Branch
- **Definition:** BRANCH_MULTIWAY.TAKEN.CORRECT_PREDICTIONS is analogous to BRANCH_MULTIWAY.ALL_PATHS.CORRECT_PREDICTIONS, except it applies only to multiway branch bundles where at least one branch instruction is taken
- **Event Code:** 0x0E, **Umask:** 1101, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_MULTIWAY.TAKEN.WRONG_PATH

- **Title:** Incorrect Predicate Predictions on Taken Multiway Bundles, **Category:** Branch
- **Definition:** BRANCH_MULTIWAY.TAKEN.WRONG_PATH is analogous to BRANCH_MULTIWAY.ALL_PATHS.WRONG_PATH, except it applies only to multi-way branch bundles where at least one branch instruction is taken
- **Event Code:** 0x0E, **Umask:** 1110, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_MULTIWAY.TAKEN.WRONG_TARGET

- **Title:** Incorrect Target Predictions on Taken Multiway Bundles, **Category:** Branch
- **Definition:** BRANCH_MULTIWAY.TAKEN.WRONG_TARGET should equal BRANCH_MULTIWAY.ALL_PATHS.WRONG_TARGET, since only multiway branch bundles where at least one branch instruction is taken actually specify a target
- **Event Code:** 0x0E, **Umask:** 1111, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.ALL.NT_OUTCOMES_CORRECTLY_PREDICTED

- **Title:** Correct Not-Taken Predicate Predictions, **Category:** Branch
- **Definition:** BRANCH_PATH.ALL.NT_OUTCOMES_CORRECTLY_PREDICTED counts the number of correct not-taken predicate predictions on not-taken branches, independent of predictor
- **Event Code:** 0x0F, **Umask:** 0010, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.ALL.TK_OUTCOMES_CORRECTLY_PREDICTED

- **Title:** Correct Taken Predicate Predictions, **Category:** Branch
- **Definition:** BRANCH_PATH.ALL.TK_OUTCOMES_CORRECTLY_PREDICTED counts the number of correct taken predicate predictions on taken branches, independent of predictor. Only the predicate must be correct; resteers to incorrect targets are also counted by this monitor as long as the branch is actually taken
- **Event Code:** 0x0F, **Umask:** 0011, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.ALL.NT_OUTCOMES_INCORRECTLY_PREDICTED

- **Title:** Incorrect Taken Predicate Predictions, **Category:** Branch
- **Definition:** BRANCH_PATH.ALL.NT_OUTCOMES_INCORRECTLY_PREDICTED counts the number of incorrect taken predicate predictions on not-taken branches, independent of predictor
- **Event Code:** 0x0F, **Umask:** 0000, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.ALL.TK_OUTCOMES_INCORRECTLY_PREDICTED

- **Title:** Incorrect Not-Taken Predicate Predictions, **Category:** Branch
- **Definition:** BRANCH_PATH.ALL.TK_OUTCOMES_INCORRECTLY_PREDICTED counts the number of incorrect not-taken predicate predictions on taken branches, independent of predictor
- **Event Code:** 0x0F, **Umask:** 0001, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.1ST_STAGE.NT_OUTCOMES_CORRECTLY_PREDICTED

- **Title:** Correct Not-Taken Predicate Predictions made in the first pipeline stage,

  Category: Branch
- **Definition:** BRANCH_PATH.1ST_STAGE.NT_OUTCOMES_CORRECTLY_PREDICTED should always count zero, as the TAR is the only predictor in the first stage of the core pipeline and it only makes taken predictions
- **Event Code:** 0x0F, **Umask:** 0110, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.1ST_STAGE.TK_OUTCOMES_CORRECTLY_PREDICTED

- **Title:** Correct Taken Predicate Predictions made in the first pipeline stage, **Category:**

  Branch
- **Definition:** BRANCH_PATH.1ST_STAGE.TK_OUTCOMES_CORRECTLY_PREDICTED counts the number of correct taken predicate predictions on taken branches made by the TAR in the first stage of the core pipeline. Only the predicate must be correct; resteers to incorrect targets are also counted by this monitor as long as the branch is actually taken. There are 0 bubbles between the branch and its predicted target
- **Event Code:** 0x0F, **Umask:** 0111, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.1ST_STAGE.NT_OUTCOMES_INCORRECTLY_PREDICTED

- **Title:** Incorrect Taken Predicate Predictions made in the first pipeline stage, **Category:**

  Branch
- **Definition:** BRANCH_PATH.1ST_STAGE.NT_OUTCOMES_INCORRECTLY_PREDICTED counts the number of incorrect taken predicate predictions on not-taken branches, made by the TAR in the first stage of the core pipeline
- **Event Code:** 0x0F, **Umask:** 0100, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.1ST_STAGE.TK_OUTCOMES_INCORRECTLY_PREDICTED

- **Title:** Incorrect Taken Predicate Predictions made in the first pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PATH.1ST_STAGE.TK_OUTCOMES_INCORRECTLY_PREDICTED should always count zero, as the TAR is the only predictor in the first stage of the core pipeline and it only makes taken predictions
- **Event Code:** 0x0F, **Umask:** 0101, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.2ND_STAGE.NT_OUTCOMES_CORRECTLY_PREDICTED

- **Title:** Correct Taken Predicate Predictions made in the second pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PATH.2ND_STAGE.NT_OUTCOMES_CORRECTLY_PREDICTED counts the number of correct not-taken predicate predictions on not-taken branches made by the BPT/MBPT in the second stage of the core pipeline
- **Event Code:** 0x0F, **Umask:** 1010, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.2ND_STAGE.TK_OUTCOMES_CORRECTLY_PREDICTED

- **Title:** Correct Taken Predicate Predictions made in the second pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PATH.2ND_STAGE.TK_OUTCOMES_CORRECTLY_PREDICTED counts the number of correct taken predicate predictions on taken branches by the BPT/MBPT or the TAC in the second stage of the core pipeline. Only the predicate must be correct; resteers to incorrect targets are also counted by this monitor as long as the branch is actually taken. There is 1 bubble between the branch and its predicted target
- **Event Code:** 0x0F, **Umask:** 1011, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.2ND_STAGE.NT_OUTCOMES_INCORRECTLY_PREDICTED

- **Title: Incorrect Taken Predicate Predictions made in the second pipeline stage**,

  Category: Branch

- **Definition:** BRANCH_PATH.2ND_STAGE.NT_OUTCOMES_INCORRECTLY_PREDICTED counts the number of incorrect taken predicate predictions on not-taken branches made by the BPT/MBPT or the TAC in the second stage of the core pipeline
- **Event Code:** 0x0F, **Umask:** 1000, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.2ND_STAGE.TK_OUTCOMES_INCORRECTLY_PREDICTED

- **Title:** Incorrect Not-Taken Predicate Predictions made in the second pipeline stage,

  Category: Branch

- **Definition:** BRANCH_PATH.2ND_STAGE.TK_OUTCOMES_INCORRECTLY_PREDICTED counts the number of incorrect not-taken predicate predictions on taken branches made by the BPT/MBPT in the second stage of the core pipeline
- **Event Code:** 0x0F, **Umask:** 1001, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.3RD_STAGE.NT_OUTCOMES_CORRECTLY_PREDICTED

- **Title:** Correct Not-Taken Predicate Predictions made in the third pipeline stage,

  Category: Branch

- **Definition:** BRANCH_PATH.3RD_STAGE.NT_OUTCOMES_CORRECTLY_PREDICTED counts the number of correct not-taken predicate predictions on not-taken branches made by the BAC in the third stage of the core pipeline, including overrides of TAR taken predictions (made in the first stage) on the last instances of loop-closing branches
- **Event Code:** 0x0F, **Umask:** 1110, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.3RD_STAGE.TK_OUTCOMES_CORRECTLY_PREDICTED

- **Title:** Correct Taken Predicate Predictions made in the third pipeline stage, **Category:**

  Branch

- **Definition:** BRANCH_PATH.3RD_STAGE.TK_OUTCOMES_CORRECTLY_PREDICTED counts the number of correct taken predicate predictions on taken branches made by the BAC in the third stage of the core pipeline. Only the predicate must be correct; resteers to incorrect targets are also counted by this monitor as long as the branch is actually taken. There are 2 bubbles between the branch and its predicted target (or 3, if the target must be computed for a branch syllable in slot 0 or 1)
- **Event Code:** 0x0F, **Umask:** 1111, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.3RD_STAGE.NT_OUTCOMES_INCORRECTLY_PREDICTED

- **Title:** Incorrect Taken Predicate Predictions made in the third pipeline stage, **Category:**

  Branch

- **Definition:** BRANCH_PATH.3RD_STAGE.NT_OUTCOMES_INCORRECTLY_PREDICTED counts the number of incorrect taken predicate predictions on not-taken branches made by the BAC in the third stage of the core pipeline
- **Event Code:** 0x0F, **Umask:** 1100, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PATH.3RD_STAGE.TK_OUTCOMES_INCORRECTLY_PREDICTED

- **Title:** Incorrect Not-Taken Predicate Predictions made in the third pipeline stage,

  Category: Branch

- **Definition:** BRANCH_PATH.3RD_STAGE.TK_OUTCOMES_INCORRECTLY_PREDICTED counts the number of incorrect not-taken predicate predictions on taken branches made by the BAC in the third stage of the core pipeline, including overrides of TAR taken predictions (made in the first stage) on the last instances of loop-closing branches
- **Event Code:** 0x0F, **Umask:** 1101, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.ALL.ALL_PREDICTIONS

- **Title:** All Branch Predictions, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.ALL.ALL_PREDICTIONS counts all branch predictions, which take place in the front-end of the processor. Note that this number does not necessarily equal the total number of branch instructions in the code, as branch predictions are made on a bundle basis (i.e., there is only one prediction per multiway branch bundle)
- **Event Code:** 0x10, **Umask:** 0000, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.ALL.CORRECT_PREDICTIONS

- **Title:** Correct Branch Predictions by All Predictors, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.ALL.CORRECT_PREDICTIONS counts all branch predictions that do not necessitate a back-end branch misprediction flush, independent of predictor. A mismatch between the predicted and actual values of the branch predicate or target results in a branch misprediction. Return branches must additionally predict privilege level and previous function state
- **Event Code:** 0x10, **Umask:** 0001, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.ALL.WRONG_PATH

- **Title:** Incorrect Predicate Predictions by All Predictors, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.ALL.WRONG_PATH counts branch mispredictions that result from a mismatch of the predicted and actual values of the branch predicate, independent of predictor
- **Event Code:** 0x10, **Umask:** 0010, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.ALL.WRONG_TARGET

- **Title:** Incorrect Target Predictions by All Predictors, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.ALL.WRONG_TARGET counts branch mispredictions that result from a mismatch of the predicted and actual values of the branch target, independent of predictor
- **Event Code:** 0x10, **Umask:** 0011, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.1ST_STAGE.ALL_PREDICTIONS

- **Title:** All Branch Predictions made in the first pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.1ST_STAGE.ALL_PREDICTIONS counts the number of branch predictions made in the first stage of the core pipeline by the TAR. The TAR is the only predictor operating in that stage of the pipeline and it only makes taken predictions. The PLP in the third stage may override a TAR predicate prediction on a loop-closing branch. The prediction flow is as follows:

```
if (TAR Hit)
    monitor++
    Read Target from TAR
```

- **Event Code:** 0x10, **Umask:** 0100, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.1ST_STAGE.CORRECT_PREDICTIONS

- **Title:** Correct Branch Predictions made in the first pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.1ST_STAGE.CORRECT_PREDICTIONS counts the number of branches correctly predicted taken by the TAR, both in predicate and target
- **Event Code:** 0x10, **Umask:** 0101, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.1ST_STAGE.WRONG_PATH

- **Title:** Incorrect Predicate Predictions made in the first pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.1ST_STAGE.WRONG_PATH counts the number of actually not-taken branches predicted by the TAR (excluding overrides by the PLP)
- **Event Code:** 0x10, **Umask:** 0110, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.1ST_STAGE.WRONG_TARGET

- **Title:** Incorrect Target Predictions made in the first pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.1ST_STAGE.WRONG_TARGET counts the number of taken branches that were resteered to an incorrect target by the TAR
- **Event Code:** 0x10, **Umask:** 0111, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.2ND_STAGE.ALL_PREDICTIONS

- **Title:** All Branch Predictions in the second pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.2ND_STAGE.ALL_PREDICTIONS counts the number of branch predictions made in the second stage of the core pipeline. The following structures operate in that stage: BPT and MBPT (for predicates), TAC and RSB (for targets). Predictions are made in the second stage only if no predictions were made during the first stage. Any prediction made in this stage will be counted, except when a taken predicate prediction is made by the BPT/MBPT on a non-return branch and no target is available from the TAC. The branch prediction structures interact in the following manner:

```
if ((BPT Hit) or (MBPT Hit))
    if (Predicted Taken)
        if (Predicted Return Branch)
            monitor++
            Read Target from RSB
        else
            if (TAC Hit)
                monitor++
                Read Target from TAC
            else
                Get Target from BAC in the 3rd Stage
    else
        monitor++
        Follow Sequential Path
else

    if (TAC Hit)
        monitor++
        Read Target from TAC
    else
        Follow Sequential Path
```

- **Event Code:** 0x10, **Umask:** 1000, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.2ND_STAGE.CORRECT_PREDICTIONS

- **Title:** Correct Branch Predictions made in the second pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.2ND_STAGE.CORRECT_PREDICTIONS counts the number of correct predicate predictions made by the BPT/MBPT or the TAC in the second stage of the core pipeline. If the predicate prediction is taken, the correct target must be provided during that stage by the RSB or the TAC. Correct taken predicate predictions made by the BPT/MBPT on non-return branches that miss the TAC require the BAC to provide a target in the third stage and are not counted by this monitor
- **Event Code:** 0x10, **Umask:** 1001, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.2ND_STAGE.WRONG_PATH

- **Title:** Incorrect Predicate Predictions made in the second pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.2ND_STAGE.WRONG_PATH counts the number of incorrect not-taken predicate predictions made in the second stage of the core pipeline, and the number of incorrect taken predicate predictions made in that stage if a target was also provided
- **Event Code:** 0x10, **Umask:** 1010, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.2ND_STAGE.WRONG_TARGET

- **Title:** Incorrect Target Predictions made in the second pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.2ND_STAGE.WRONG_TARGET counts the number of branches that were correctly predicted taken by the BPT/MBPT or TAC, but were resteered to an incorrect target by the RSB or the TAC
- **Event Code:** 0x10, **Umask:** 1011, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.3RD_STAGE.ALL_PREDICTIONS

- **Title:** All Branch Predictions made in the third pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.3RD_STAGE.ALL_PREDICTIONS counts the number of branch predictions made in the third stage of the core pipeline by the BAC. The BAC can make both predicate predictions (based on the whether hint field of the branch) and target predictions, in the following manner:

```
if (TAR Hit)
    if (Predicted Last Instance of Loop-Closing Branch)
        monitor++
        PLP Override of TAR Taken Prediction
        Resteer Frontend to Sequential Address
else
    if ((BPT Hit) or (MBPT Hit))
        if (Predicted Taken)
            if (not (TAC Hit))
                if (not (Predicted Return Branch))
                    monitor++
                    Compute Target
    else
        if (not (TAC Hit))
            monitor++
            Read Whether Hint Field for Predicate Prediction
            if (Predicted Taken)
                Read BType Field for Type Information
                if (Indirect Branch)
                    Read Target from RSB
                else
```

```
            Compute Target
    else
        Follow Sequential Path
```

- **Event Code:** 0x10, **Umask:** 1100, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.3RD_STAGE.CORRECT_PREDICTIONS

- **Title:** Correct Branch Predictions made in the third pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.3RD_STAGE.CORRECT_PREDICTIONS counts the number of correct branch predictions made by the BAC, including target predictions of branches whose predicate was supplied by a different predictor. For predicted-taken branches, both predicate and target must be correct
- **Event Code:** 0x10, **Umask:** 1101, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.3RD_STAGE.WRONG_PATH

- **Title:** Incorrect Predicate Predictions made in the third pipeline stage, **Category:**

  Branch
- **Definition:** BRANCH_PREDICTOR.3RD_STAGE.WRONG_PATH counts branches whose predicate was incorrectly predicted by the BAC (based on the whether hint field of the branch), and not-taken branches whose taken predicate prediction by another predictor caused the BAC to supply a target
- **Event Code:** 0x10, **Umask:** 1110, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_PREDICTOR.3RD_STAGE.WRONG_TARGET

- **Title:** Incorrect Target Predictions made in the third pipeline stage, **Category:** Branch
- **Definition:** BRANCH_PREDICTOR.3RD_STAGE.WRONG_TARGET counts taken branches that were correctly predicted taken by any predictor, but whose target was incorrectly supplied by the BAC
- **Event Code:** 0x10, **Umask:** 1111, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

BRANCH_TAKEN_SLOT

- **Title:** Taken Branch Detail, **Category:** Branch
- **Definition:** BRANCH_TAKEN_SLOT monitors which slot number in a branch bundle (single-way or multiway) a taken branch occupies, or records that there were no taken branches in the given branch bundle. Use this monitor behind the downstream opcode matcher, rather than IA64_TAGGED_INST_RETIRED, to count dynamic br.calls and br.rets.
- **Event Code:** 0x0D, **Umask:** See below, **PMC/PMD:** 4,5,6,7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

  The SLOT_MASK unit mask defined by Table 7-29 allows profiling of taken branches based on their instruction slot number. If multiple bits are set in the SLOT_MASK, all the set cases are included in the event count. The processor uses the following equation to determine the event outcome in each cycle:

```
    (PMC.umask{16}
       and (branch in slot 0 is taken))
or (PMC.umask{17}
    and (branch in slot 0 is NOT taken)
    and (branch in slot 1 is taken))
or  (PMC.umask{18}
    and (branch in slot 0 is NOT taken)
    and (branch in slot 1 is NOT taken)
    and (branch in slot 2 is taken))
or  (PMC.umask{19}
    and (branch in slot 0 is NOT taken)
    and (branch in slot 1 is NOT taken)
    and (branch in slot 2 is NOT taken))
```

Table 7-29. Slot unit mask for BRANCH_TAKEN_SLOT

SLOT_MASK	PMC.umask {19:16}	Description
Instruction Slot 0	xxx1	Count if branch in slot 0 is first taken branch
Instruction Slot 1	xx1x	Count if branch in slot 1 is first taken branch
Instruction Slot 2	x1xx	Count if branch in slot 2 is first taken branch
No taken branch	1xxx	Count if NO branch was taken

BUS_ALL

- **Title:** Bus Transactions **Category:** Frontside Bus
- **Definition:** BUS_ALL counts all transactions issued on the bus. These include BRL, BRIL, BIL, BWL, BRP, BWP, IORD, IOWR, and the other transactions.
- **Event Code:** 0x47, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_BRQ_LIVE_REQ_LO and BUS_BRQ_LIVE_REQ_HI

- **Title:** BRQ Live Requests, **Category:** Frontside Bus
- **Definition:** BUS_BRQ_LIVE_REQ counts the number of live entries in the bus request queue (BRQ). These events include L3 cache reads, BRL, BRIL, BRP, and IORD memory transactions. The count excludes cache line write backs, partial writes (BWP and IOWR) and write coalescing read for ownership transactions, since these have their own write queue. This performance monitor increments its count each core clock (not bus clock).
- **Event Code:** 0x5c (HI), 0x5b (LO), **Umask:** Ignored, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 2 (each)
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

  Two hardware events are needed to count this:

  BUS_BRQ_LIVE_REQ_HI - counts the two most significant bit of the 4-bit outstanding BRQ request count

  BUS_BRQ_LIVE_REQ_LO - counts the two least significant bit of the 4-bit outstanding BRQ request count

BUS_BRQ_READ_REQ_INSERTED

- **Title:** BRQ Requests Inserted **Category:** Frontside Bus
- **Definition:** BUS_BRQ_READ_REQ_INSERTED counts the number of reads (BRL) and read for ownership (BRIL) requests that are inserted into the BRQ. The count excludes cache line write backs, partial and coalescing writes, since these have their own write queue.
- **Event Code:** 0x5d, **Umask:** Ignored, **PMC/PMD:**4, 5, 6, 7**Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

  BRQ inserts can be used to directly measure combined far cache/BUS latencies as follows:

  avg_brq_req_outstanding_per_cycle = (BUS_BRQ_LIVE_REQ/delta(cycles))

  avg_brq_latency = (BUS_BRQ_LIVE_REQ / BUS_BRQ_READ_REQ_INSERTED)

  The only caveat is that the tracked BRQ inserts holds read and read for ownership, but not write coalescing write backs.

BUS_BURST

- **Title:** Bus Burst Transactions **Category:** Frontside Bus
- **Definition:** BUS_BURST counts the number of full cache line (burst mode) bus memory transactions. These include BRL, BRIL, and BWL transactions.
- **Event Code:** 0x49, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_HITM

- **Title:** Bus Hit Modified Line Transactions **Category:** Frontside Bus
- **Definition:** BUS_HITM counts the number of memory transactions which caused HITM to be asserted. The following memory transactions are included in the performance monitor: BRL, BWL, BRIL, and BIL. Only events originated by this processor are counted.
- **Event Code:** 0x44, **Umask:** Ignored, **PMC/PMD:**4, 5, 6, 7**Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_IO

- **Title:** IA-32 Compatible I/O Bus Transactions **Category:** Frontside Bus
- **Definition:** BUS_IO counts the number of I/O transactions. These include either IORD or IOWR transactions.
- **Event Code:** 0x50, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

Note: only the “Any” and “Self” unit masks are supported.

BUS_IOQ_LIVE_REQ_LO and BUS_IOQ_LIVE_REQ_HI

- **Title:** In-Order Bus Queue Results, **Category:** Frontside Bus
- **Definition:** BUS_IOQ_LIVE_REQ counts the number of live bus requests in the in order bus queue. This performance monitor increments its count each core clock (not bus clock).
- **Event Code:** 0x58 (HI), 0x57 (LO), **Umask:** Ignored, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 3 (each)
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

  BUS_IOQ_LIVE_REQ can then be computed as BUS_IOQ_LIVE_REQ_HI*4 + BUS_IOQ_LIVE_REQ_LO

  Two hardware events are needed to count this:

  BUS_IOQ_LIVE_REQ_HI - counts the two most significant bit write backs of the 4-bit outstanding IOQ request count

  BUS_IOQ_LIVE_REQ_LO - counts the two least significant bits of the 4-bit outstanding IOQ request count

BUS_LOCK

- **Title:** IA-32 Compatible Bus Lock Transactions **Category:** Frontside Bus
- **Definition:** BUS_LOCK counts the number of IA-32 compatible bus lock transactions.
- **Event Code:** 0x53, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

Note: Only the “Any” unit mask is supported.

BUS_LOCK_CYCLES

- **Title:** IA-32 Compatible Bus Lock Cycles **Category:** Frontside Bus
- **Definition:** BUS_LOCK_CYCLES counts the number of bus clocks that the bus is locked due to IA-32 compatible bus lock transactions.
- **Event Code:** 0x54, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

Note: Only the “Any” unit mask is supported.

BUS_MEMORY

- **Title:** Bus Memory Transactions **Category:** Frontside Bus
- **Definition:** BUS_MEMORY counts the number of bus memory transactions. These include BRL, BRIL, BIL, BWL, BRP, and BWP transactions.
- **Event Code:** 0x4a, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_PARTIAL

- **Title:** Bus Partial Transactions **Category:** Frontside Bus
- **Definition:** BUS_PARTIAL counts the number of partial bus memory transactions. These include BRP and BWP transactions.
- **Event Code:** 0x48, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_RD_ALL

- **Title:** Bus Read Transactions **Category:** Frontside Bus
- **Definition:** BUS_RD_ALL counts the number of BRL memory transactions. These include both code and data BRL transactions.
- **Event Code:** 0x4b, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_RD_DATA

- **Title:** Bus Read Data Transactions **Category:** Frontside Bus
- **Definition:** BUS_RD_DATA counts the number of BRL data transactions.
- **Event Code:** 0x4c, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_RD_HIT

- **Title:** Bus Read Hit Clean Non-local Cache Transactions **Category:** Frontside Bus
- **Definition:** BUS_RD_HIT counts the number of BRL memory transactions which caused HIT to be asserted. Only events originated by this processor are counted.
- **Event Code:** 0x40, **Umask:** Ignored, **PMC/PMD:**4, 5, 6, 7**Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_RD_HITM

- **Title:** Bus Read Hit Modified Non-local Cache Transactions **Category:** Frontside Bus
- **Definition:** BUS_RD_HITM counts the number of BRL memory transactions which caused HITM to be asserted. Only events originated by this processor are counted.
- **Event Code:** 0x41, **Umask:** Ignored, **PMC/PMD:**4, 5, 6, 7**Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_RD_INVAL

- **Title:** Bus Read Invalidated Line **Category:** Frontside Bus
- **Definition:** BUS_RD_INVAL counts the number of BIL memory transactions. On Itanium processors, these transactions are only generated from flush cache instructions.
- **Event Code:** 0x4e, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_RD_INVAL_BST

- **Title:** Bus BRIL Burst Transactions **Category:** Frontside Bus
- **Definition:** BUS_RD_INVAL counts the number of BRIL memory transactions. These transactions are typically generated from memory stores, RFO (read for ownership) events.
- **Event Code:** 0x4f, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_RD_INVAL_HITM

- **Title:** Bus BIL Transaction Results in HITM**Category:** Frontside Bus
- **Definition:** BUS_RD_INVAL_HITM counts the number of BIL transactions which caused HITM to be asserted.Only events originated by this processor are counted.
- **Event Code:** 0x42, **Umask:** Ignored, **PMC/PMD:**4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_RD_INVAL_BST_HITM

- **Title:** Bus BRIL Burst Transaction Results in HITM **Category:** Frontside Bus
- **Definition:** BUS_RD_INVAL_BST_HITM counts the number of BRIL transactions which caused HITM to be asserted. Only events originated by this processor are counted.
- **Event Code:** 0x43, **Umask:** Ignored, **PMC/PMD:**4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_RD_IO

- **Title:** IA-32 Compatible I/O Read Transactions **Category:** Frontside Bus
- **Definition:** BUS_RD_IO counts the number of IORD transactions.
- **Event Code:** 0x51, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

Note: Only the “Any” and “Self” unit masks are supported.

BUS_RD_PRTL

- **Title:** Bus Read Partial Transactions **Category:** Frontside Bus
- **Definition:** BUS_RD_PRTL counts the number of partial read memory transactions. These include BRP transactions.
- **Event Code:** 0x4d, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_SNOOPS

- **Title:** Bus Snoops Total **Category:** Frontside Bus
- **Definition:** BUS_SNOOPS counts the number of internal snoops generated from bus memory transactions.
- **Event Code:** 0x46, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

Note: Only the "Any" unit mask is supported. "Any" counts the number of internal snoops generated from all bus transactions.

BUS_SNOOPS_HITM

- **Title:** Bus Snoops Hit Modified Cache Line **Category:** Frontside Bus
- **Definition:** BUS_SNOOPS_HITM counts the number of internal snoops (generated from bus memory transactions) which hit a modified line in the local processor.
- **Event Code:** 0x45, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

Note: Only the "Any" mask is supported. "Any" counts the number of internal snoops, generated from all transactions, which hit a modified line.

BUS_SNOOP_STALL_CYCLES

- **Title:** Bus Snoop Stall Cycles **Category:** Frontside Bus
- **Definition:** BUS_SNOOP_STALL_CYCLES counts the number of bus clocks that the bus is stalled due to snoop stalls.
- **Event Code:** 0x55, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

Note: Only "Self" and "Any" unit masks are supported. "Self" counts the number of snoop stalls generated due to memory transactions initiated by the local processor. "Any" counts all snoop stalls (those generated due to memory transactions initiated by the local processor, other processors, and the priority agent).

BUS_SNOOPQ_REQ

- **Title:** Bus Snoop Queue Requests, **Category**: Frontside Bus
- **Definition:** BUS_SNOOPQ_REQ counts the number of outstanding memory transactions that have not completed the snoop phase. This performance monitor increments its count each core clock (not bus clock). BUS_SNOOPQ_REQ is not equivalent to the number of valid entries in the snoop queue. This is due to the fact that entries can stay in the snoop queue beyond the snoop phase (e.g. for implicit write backs).
- **Event Code:** 0x56, **Umask:** Ignored, **PMC/PMD:**4, 5, **Max. Increment/Cycle:** 3
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

BUS_WR_WB

- **Title:** Bus Write Back Transactions **Category:** Frontside Bus
- **Definition:** BUS_WR_WB counts the number of BWL memory transactions. These transactions are generated from either explicit write backs or coalescing writes. Currently, these will count BWL (if snoops are disabled).
- **Event Code:** 0x52, **Umask:** See Section 7.6.5, **PMC/PMD:**4, 5, 6, 7

  Max. Increment/Cycle: 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

CPU_CPL_CHANGES

- **Title:** Privilege level changes, **Category:** System
- **Definition:** CPU_CPL_CHANGES counts the number of privilege level changes
- **Event Code:** 0x34, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

CPU_CYCLES

- **Title:** CPU Cycles, **Category:** System
- **Definition:** CPU_CYCLES counts elapsed processor cycles
- **Event Code:** 0x12, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

DATA_ACCESS_CYCLE

- **Title:** Data Access Stall Cycles, **Category:** Stall
- **Definition:** DATA_ACCESS_CYCLE counts the number of cycles that the pipeline is stalled or flushed due to instructions waiting for data on cache misses, L1D way mispredictions, and DTC misses.
- **Event Code:** 0x03, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

DATA_EAR_EVENTS

- **Title:** L1 Data Cache EAR Events, **Category:** L1 Data Cache
- **Definition:** DATA_EAR_EVENTS counts the number of data cache or DTLB events captured by the Data Cache Unit Event Address Register
- **Event Code:** 0x67, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: no

DATA_REFERENCES_RETIRED

- **Title:** Retired Data Memory References, **Category:** L1 Data Cache
- **Definition:** DATA_REFERENCES_RETIRED counts the number of data memory references retired by the processor memory pipeline. The count includes check loads, uncacheable accesses, RSE operations, VHPT memory references, semaphores, and FP memory references. Predicated off operations are excluded
- **Event Code:** 0x63, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: yes

DEPENDENCY_ALL_CYCLE

- **Title:** Scoreboard Dependency and Dispersal Break Cycles, **Category:** Stall
- **Definition:** DEPENDENCY_ALL_CYCLE counts the number of cycles attributable to data (scoreboard) dependency on integer or FP operations (not counting cache/memory access), or issue-limit stalls (e.g., implicit and explicit stops). Floating-point flushes and delays due to control and application register reads and writes are factored in as well.
- **Event Code:** 0x06, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

DEPENDENCY_SCOREBOARD_CYCLE

- **Title:** Scoreboard Dependency Cycles, **Category:** Stall
- **Definition:** DEPENDENCY_SCOREBOARD_CYCLE counts the number of cycles attributable to data (scoreboard) dependency on integer or FP operations (not counting cache/memory access). Floating-point flushes and delays due to control and application register reads and writes are factored in as well.
- **Event Code:** 0x02, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

DTC_MISSES

- **Title:** DTC Misses, **Category:** System
- **Definition:** DTC_MISSES counts the number of DTC misses for data requests
- **Event Code:** 0x60, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: no

DTLB_INSERTS_HPW

- **Title:** Hardware Page Walker Inserts into the DTLB, **Category:** System
- **Definition:** DTLB_INSERTS_HPW counts the number of DTLB inserts completed by the hardware page table walker
- **Event Code:** 0x62, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: no

DTLB_MISSES

- **Title:** DTLB Misses, **Category:** System
- **Definition:** DTLB_MISSES counts the number of DTLB misses for demand requests
- **Event Code:** 0x61, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: no

EXPL_STOPS_DISPERSED

- **Title:** Explicit Stops Dispersed, **Category:** Instruction Issue
- **Definition:** EXPL_STOPS_DISPERSED counts the number of explicit programmer-specified stops, including those encountered during hardware speculative wrong-path execution
- **Event Code:** 0x2E, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

FP_OPS_RETIRED_HI

- **Title:** FP Operations Retired (High), **Category:** Execution
- **Definition:** FP_OPS_RETIRED_HI and FP_OPS_RETIRED_LO together compute the derived event FP_OPS_RETIRED.d which is the weighted sum of retired FP operations
- **Event Code:** 0x0A, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 3
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: no

  FP_OPS_RETIRED.d, a derived value, is computed as FP_OPS_RETIRED_HI * 4 + FP_OPS_RETIRED_LO. Weights for individual FP ops: `fnorm`=1, `fadd`=1, `fmpy=1`, `fma`=2, `fms`=2, `fsub`=1, `fpma`=4, `fpmpy`=4, `fpms`=4, `fnma`=2, `frcpa`=1, `frsqrta`=1, `fpnma`=4, `fprcpa`=2, `fprsqrta`=2, `xma`=0

Note: Integer multiply instructions (xma) are not counted as floating-point operations (even though they are executed in the floating-point multiplier).

FP_OPS_RETIRED_LO

- **Title:** FP Operations Retired (Low), **Category:** Execution
- **Definition:** FP_OPS_RETIRED_HI and FP_OPS_RETIRED_LO together compute the derived event FP_OPS_RETIRED.d which is the weighted sum of retired FP operations
- **Event Code:** 0x09, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 3
- **Qualification:** See FP_OPS_RETIRED_HI on page 7-44

FP_FLUSH_TO_ZERO

- **Title:** FP Result Flushed to Zero, **Category:** Execution
- **Definition:** FP_FLUSH_TO_ZERO counts the number of times a near zero result is flushed to zero in FTZ mode. Parallel FP operations which cause one or both results to flush to zero will increment the event count only by one (i.e. even if both results are flushed to zero)
- **Event Code:** 0x0B, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: no

FP_SIR_FLUSH

- **Title:** FP SIR Flushes, **Category:** Execution
- **Definition:** FP_SIR_FLUSH counts the number of times a Safe Instruction Recognition (SIR) flush occurs
- **Event Code:** 0x0C, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: no

IA32_INST_RETIRED

- **Title:** Retired IA-32 Instructions, **Category:** System
- **Definition:** IA32_INST_RETIRED counts the number of IA-32 instructions retired
- **Event Code:** 0x15, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

IA64_INST_RETIRED

- **Title:** Retired IA-64 Instructions, **Category:** Execution
- **Definition:** IA64_INST_RETIRED counts all retired IA-64 instructions. The count includes predicated on and off instructions, NOPs, but excludes hardware-inserted RSE operations. This event is equal to IA64_TAGGED_INST_RETIRED with a zero unit mask
- **Event Code:** 0x08, **Umask:** 0000, **PMC/PMD:** 4, 5 **Max. Increment/Cycle:** 6
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: no

IA64_TAGGED_INST_RETIRED

- **Title:** Retired Tagged IA-64 Instructions, **Category:** Execution
- **Definition:** IA64_TAGGED_INST_RETIRED is analogous to IA64_INST_RETIRED, except that it further qualifies event selection with the instruction address range and opcode match settings in the IBR and PMC registers
- **Event Code:** 0x08, **Umask:** See below, **PMC/PMD:** 4, 5 **Max. Increment/Cycle:** 6
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: no

  The TAG_SELECT unit mask defined in Table 7-30 always qualifies the event count of IA64_TAGGED_INST_RETIRED with either the opcode match register PMC_8 or PMC_9. Note that the setting of PMC_8 qualifies all down-stream event monitors. To ensure that other monitored events are counted independent of the opcode matcher, all mifb and all mask bits of PMC_8 should be set to one (all opcodes match). The settings of PMC9 do not affect other event monitors.

  Also, note that umask 0011 is distinct in that it also counts, in addition to instructions matched by the appropriate opode matcher, architecturally invisible RSE fills and spills when the parent instruction (such as an alloc or br.ret) causing them is matched by the combination in PMC8. Thus, the difference in counts obtained between using PMC8 and PMC9 as opcode matchers is the amount of RSE activity.

Table 7-30. Retired Event Selection by Opcode Match

TAG_SELECT	PMC.umask {19:16}	Description
PMC_8 tag	0011	Instruction tagged by opcode matcher PMC_8
PMC_9 tag	0010	Instruction tagged by opcode matcher PMC_9
All	0000	All retired instructions (regardless of whether they were tagged or not)
Undefined	All other umask settings	Undefined event count

INST_ACCESS_CYCLE

- **Title:** Instruction Access Cycles, **Category:** Stall
- **Definition:** INST_ACCESS_CYCLE counts the number of cycles where there are no back-end stalls or flushes, the decoupling buffer is empty, and the front-end is stalled waiting on an L1I or ITLB miss.
- **Event Code:** 0x01, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

INST_DISPERSED

- **Title:** Instructions Dispersed, **Category:** Instruction Issue
- **Definition:** INST_DISPERSED counts the number of instructions dispersed (including nops) from the front-end to the back-end of the machine. The count includes instruction dispersal on the wrong execution path; i.e., in the shadow of a branch misprediction flush or other back-end flush
- **Event Code:** 0x2D, **Umask:** Ignored, **PMC/PMD:** 4, 5 **Max. Increment/Cycle:** 6
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

INST_FAILED_CHKS_RETIRED.ALL

- **Title:** Failed Speculative Check Loads, **Category:** Execution
- **Definition:** INST_FAILED_CHKS_RETIRED.ALL counts the number of failed speculative check load instructions (`chk.s`). The count excludes predicated off `chk.s` instructions and includes both integer and FP variants
- **Event Code:** 0x35, **Umask:** xx11, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: no

INST_FAILED_CHKS_RETIRED.FP

- **Title:** Failed Speculative FP Check Loads, **Category:** Execution
- **Definition:** INST_FAILED_CHKS_RETIRED.FP counts the number of failed speculative check load instructions (`chk.s`). The count excludes predicated off `chk.s` instructions and includes only FP variants
- **Event Code:** 0x35, **Umask:** xx10, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: no

INST_FAILED_CHKS_RETIRED.INTEGER

- **Title:** Failed Speculative Integer Check Loads, **Category:** Execution
- **Definition:** INST_FAILED_CHKS_RETIRED.INTEGER counts the number of failed speculative check load instructions (`chk.s`). The count excludes predicated off `chk.s` instructions and includes only integer variants
- **Event Code:** 0x35, **Umask:** xx01, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: no
- **Qualification:**

INSTRUCTION_EAR_EVENTS

- **Title:** Instruction EAR Events, **Category:** Instruction Cache
- **Definition:** INSTRUCTION_EAR_EVENTS counts the number of EAR captures for L1I and ITLB events
- **Event Code:** 0x23, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

ISA_TRANSITIONS

- **Title:** IA-64 to IA-32 ISA Transitions, **Category:** System
- **Definition:** ISA_TRANSITIONS counts the number of instruction set transitions from IA-64 to IA-32. This is the number of times the PSR.is bit toggles from 0 to 1 due to `br.ia` or `rfi` to IA-32 code
- **Event Code:** 0x14, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

ISB_LINES_IN

- **Title:** Instruction Streaming Buffer Lines In, **Category:** Instruction Cache
- **Definition:** ISB_LINES_IN counts the number of 32-byte L1I cache lines written from L2 (and beyond) into the Instruction Streaming Buffer as a consequence of instruction demand miss and instruction prefetch requests
- **Event Code:** 0x26, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

ITLB_INSERTS_HPW

- **Title:** Hardware Page Walker Inserts into the ITLB, **Category:** System
- **Definition:** ITLB_INSERTS_HPW counts the number of ITLB inserts done by the hardware page table walker
- **Event Code:** 0x28, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

ITLB_MISSES_FETCH

- **Title:** ITLB Demand Misses, **Category:** System
- **Definition:** ITLB_MISSES_FETCH counts the number of demand ITLB misses
- **Event Code:** 0x27, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

L1D_READ_FORCED_MISSES_RETIRED

- **Title:** L1 Data Cache Forced Load Misses, **Category:** L1 Data Cache
- **Definition:** L1D_READ_FORCED_MISSES_RETIRED counts the number of loads that were forced to miss the L1 data cache due to memory ordering constraints, predicted L1 data cache misses, Store Buffer hits, or simultaneous L2 data returns to the register file
- **Event Code:** 0x6B, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: yes

L1D_READ_MISSES_RETIRED

- **Title:** L1 Data Cache Read Misses, **Category:** L1 Data Cache
- **Definition:** L1D_READ_MISSES_RETIRED counts the number of committed L1 data cache read misses. The count includes any read reference that could have been serviced by the L1 data cache (see L1D_READS_RETIRED event for a detailed list) but missed the cache. False misses are included in the event count. Since the L1 data cache is write-through, write misses are NOT counted
- **Event Code:** 0x66, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: yes

L1D_READS_RETIRED

- **Title:** L1 Data Cache Reads, **Category:** L1 Data Cache
- **Definition:** L1D_READS_RETIRED counts the number of committed L1 data cache reads (integer and RSE references). Excluded from the count are VHPT loads, check loads, L1 hinted loads, semaphores, uncacheable and FP loads. Predicated-off loads are also excluded, but wrong-path operations are included in the count
- **Event Code:** 0x64, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: yes

L1I_DEMAND_READS

- **Title:** L1I and ISB Instruction Demand Lookups, **Category:** Instruction Cache
- **Definition:** L1I_DEMAND_READS counts the number of 32-byte instruction demand L1I/ISB lookups, independent of the hit/miss outcome
- **Event Code:** 0x20, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

  Qualifications based on instruction address range may be inaccurate

L1I_FILLS

- **Title:** L1 Instruction Cache Fills, **Category:** Instruction Cache
- **Definition:** L1I_FILLS counts the number of 32-byte lines moved from the Instruction Streaming Buffer into the L1 instruction cache
- **Event Code:** 0x21, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L1I_PREFETCH_READS

- **Title:** L1I and ISB Instruction Prefetch Lookups, **Category:** Instruction Cache
- **Definition:** L1I_PREFETCH_READS counts the number of 64-byte instruction prefetch L1I/ISB lookups, independent of the hit/miss outcome.
- **Event Code:** 0x24, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

L2_DATA_REFERENCES.ALL

- **Title:** L2 Data Read and Write References, **Category:** L2 Cache
- **Definition:** L2_DATA_REFERENCES.ALL counts all L2 data read and write accesses.The reported count is the number of requests prior to cache line merging. Semaphore operations are counted as one read and one write
- **Event Code:** 0x69, **Umask:** xx11, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: yes

L2_DATA_REFERENCES.READS

- **Title:** L2 Data Read References, **Category:** L2 Cache
- **Definition:** L2_DATA_REFERENCES.READS counts all L2 data read accesses. The reported count is the number of requests prior to cache line merging. Semaphore operations are counted as one read
- **Event Code:** 0x69, **Umask:** xx01, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: yes

L2_DATA_REFERENCES.WRITES

- **Title:** L2 Data Write References, **Category:** L2 Cache
- **Definition:** L2_DATA_REFERENCES.WRITES counts all L2 data write accesses. The reported count is the number of requests prior to cache line merging. Semaphore operations are counted as one write
- **Event Code:** 0x69, **Umask:** xx10, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode matching: yes, Data Address Range: yes

L2_FLUSH_DETAILS

- **Title:** L2 Flush Details, **Category:** L2 Cache
- **Definition:** L2_FLUSH_DETAILS allows a detailed breakdown of L2 pipeline flushes by cause. This event counts the number of L2 pipeline flushes constrained by the conditions specified in the 4-bit unit mask defined by Table 7-31 on page 7-51. All combinations of the four unit mask bits are supported
- **Event Code:** 0x77, **Umask:See below, PMC/PMD:** 4, 5, 6, 7

Max. Increment/Cycle: 1

- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

Table 7-31. Unit Mask Bits {19:16} for L2_FLUSH_DETAILS Event

L2 Flush Reason	PMC.umask {19:16}	Description
L2_ST_BUFFER_FLUSH	xxx1	L2 store to store conflict due to (a) Same store buffer entry (b) Back to back stores
L2_ADDR_CONFLICT	xx1x	L2 flushed due to MESI update on load follows store
L2_BUS_REJECT	x1xx	L2 flushed due to bus constraints
L2_FULL_FLUSH	1xxx	L2 flushed due to one of: (a) Store buffer full (b) Load miss buffer full

L2_FLUSHES

- **Title:** L2 Flushes, **Category:** L2 Cache
- **Definition:** L2_FLUSHES counts the number of L2 pipeline flushes due to Store Buffer conflicts, address conflicts, full L3 and bus queues, and other such reasons
- **Event Code:** 0x76, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

L2_INST_DEMAND_READS

- **Title:** L2 Instruction Demand Fetch Requests, **Category:** Instruction Cache
- **Definition:** L2_INST_DEMAND_READS counts the number of L2 instruction requests due to L1I demand fetch misses. The monitor counts the number of demand fetch look-ups that miss in both the L1I and the ISB, regardless of whether they hit or miss in the Request Address Buffer (RAB); i.e., the count includes misses to a line that has already been requested (secondary misses)
- **Event Code:** 0x22, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

L2_INST_PREFETCH_READS

- **Title:** L2 Instruction Prefetch Requests, **Category:** Instruction Cache
- **Definition:** L2_INST_PREFETCH_READS counts all instruction prefetch requests issued to the unified L2 cache
- **Event Code:** 0x25, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

L2_MISSES

- **Title:** L2 Misses, **Category:** L2 Cache
- **Definition:** L2_MISSES counts the number of L2 cache misses (requests to uncacheable pages are excluded). The count includes misses caused by instruction fetch and prefetch, and data read and write operations. Secondary misses to the same L2 cache line will be counted as individual misses
- **Event Code:** 0x6A, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

L2_REFERENCES

- **Title:** L2 References, **Category:** L2 Cache
- **Definition:** L2_REFERENCES counts the number of L2 cache references (requests to uncacheable pages are excluded). The count includes references by instruction fetch and prefetch, and data reads and writes. The maximum per-cycle increment is three: one instruction fetch and two data references
- **Event Code:** 0x68, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 3
- **Qualification:** Instruction Address Range: yes, Opcode matching: no, Data Address Range: no

L3_LINES_REPLACED

- **Title:** L3 Cache Lines Replaced, **Category:** L3 Cache
- **Definition:** L3_LINES_REPLACED counts the number of valid L3 lines that have been victimized
- **Event Code:** 0x7F, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_MISSES

- **Title:** L3 Misses, **Category:** L3 Cache
- **Definition:** L3_MISSES counts the number of L3 misses. The number includes misses caused by both instruction and data requests and L2 line writebacks
- **Event Code:** 0x7C, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_READS.ALL_READS.ALL

- **Title:** Instruction and Data L3 Reads, **Category:** L3 Cache
- **Definition:** L3_READS.ALL_READS.ALL counts the number of all L3 read accesses, independent of the stream source (instruction or data) and the hit/miss outcome
- **Event Code:** 0x7D, **Umask:** 1111, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_READS.ALL_READS.HIT

- **Title:** Instruction and Data L3 Read Hits, **Category:** L3 Cache
- **Definition:** L3_READS.ALL_READS.HIT counts the number of all L3 read hits, independent of the stream source (instruction or data)
- **Event Code:** 0x7D, **Umask:** 1101, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_READS.ALL_READS.MISS

- **Title:** Instruction and Data L3 Read Misses, **Category:** L3 Cache
- **Definition:** L3_READS.ALL_READS.MISS counts the number of all L3 read misses, independent of the stream source (instruction or data)
- **Event Code:** 0x7D, **Umask:** 1110, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_READS.DATA_READS.ALL

- **Title:** Data L3 Reads, **Category:** L3 Cache
- **Definition:** L3_READS.DATA_READS.ALL counts the number of data L3 read accesses, independent of the hit/miss outcome
- **Event Code:** 0x7D, **Umask:** 1011, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_READS.DATA_READS.HIT

- **Title:** Data L3 Read Hits, **Category:** L3 Cache
- **Definition:** L3_READS.DATA_READS.HIT counts the number of data L3 read hits
- **Event Code:** 0x7D, **Umask:** 1001, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_READS.DATA_READS.MISS

- **Title:** Data L3 Read Misses, **Category:** L3 Cache
- **Definition:** L3_READS.DATA_READS.MISS counts the number of data L3 read misses
- **Event Code:** 0x7D, **Umask:** 1010, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_READS.INST_READS.ALL

- **Title:** Instruction L3 Reads, **Category:** L3 Cache
- **Definition:** L3_READS.INST_READS.ALL counts the number of instruction L3 read accesses, independent of the hit/miss outcome
- **Event Code:** 0x7D, **Umask:** 0111, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_READS.INST_READS.HIT

- **Title:** Instruction L3 Read Hits, **Category:** L3 Cache
- **Definition:** L3_READS.INST_READS.HIT counts the number of instruction L3 read hits
- **Event Code:** 0x7D, **Umask:** 0101, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_READS.INST_READS.MISS

- **Title:** Instruction L3 Read Misses, **Category:** L3 Cache
- **Definition:** L3_READS.INST_READS.MISS counts the number of instruction L3 read misses
- **Event Code:** 0x7D, **Umask:** 0110, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_REFERENCES

- **Title:** L3 References, **Category:** L3 Cache
- **Definition:** L3_REFERENCES counts the number of L3 cache references (requests to uncacheable pages are excluded). The count includes references by instruction fetch and prefetch, data reads and writes, and L2 cache line most significant bit writebacks.
- **Event Code:** 0x7B, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_WRITES.ALL_WRITES.ALL

- **Title:** L3 Writes, **Category:** L3 Cache
- **Definition:** L3_WRITES.ALL_WRITES.ALL counts the number of L3 write accesses independent of the hit/miss outcome. The count includes both data writes and L2 write-back accesses (including L3 read for ownership requests that satisfy stores)
- **Event Code:** 0x7E, **Umask:** 1111, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_WRITES.ALL_WRITES.HIT

- **Title:** L3 Write Hits, **Category:** L3 Cache
- **Definition:** L3_WRITES.ALL_WRITES.HIT counts the number of L3 write hits. The count includes both data writes and L2 writeback accesses (including L3 read for ownership requests that satisfy stores)
- **Event Code:** 0x7E, **Umask:** 1101, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_WRITES.ALL_WRITES.MISS

- **Title:** L3 Write Misses, **Category:** L3 Cache
- **Definition:** L3_WRITES.ALL_WRITES.MISS counts the number of L3 write misses. The count includes both data writes and L2 writeback accesses (including L3 read for ownership requests that satisfy stores)
- **Event Code:** 0x7E, **Umask:** 1110, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_WRITES.L2_WRITEBACK.ALL

- **Title:** L3 Writebacks, **Category:** L3 Cache
- **Definition:** L3_WRITES.L2_WRITEBACK.ALL counts the number of L3 write accesses that result from L2 writebacks, independent of hit/miss outcome
- **Event Code:** 0x7E, **Umask:** 1011, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_WRITES.L2_WRITEBACK.HIT

- **Title:** L3 Writeback Hits, **Category:** L3 Cache
- **Definition:** L3_WRITES.L2_WRITEBACK.HIT counts the number of L3 write hits that result from L2 writebacks
- **Event Code:** 0x7E, **Umask:** 1001, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_WRITES.L2_WRITEBACK.MISS

- **Title:** L3 Writeback Misses, **Category:** L3 Cache
- **Definition:** L3_WRITES.L2_WRITEBACK.MISS counts the number of L3 write misses that result from L2 writebacks
- **Event Code:** 0x7E, **Umask:** 1010, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_WRITES.DATA_WRITES.ALL

- **Title:** L3 Data Writes, **Category:** L3 Cache
- **Definition:** L3_WRITES.DATA_WRITES.ALL counts the number of L3 data write accesses independent of the hit/miss outcome
- **Event Code:** 0x7E, **Umask:** 0111, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_WRITES.DATA_WRITES.HIT

- **Title:** L3 Data Write Hits, **Category:** L3 Cache
- **Definition:** L3_WRITES.DATA_WRITES.HIT counts the number of L3 data write hits
- **Event Code:** 0x7E, **Umask:** 0101, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

L3_WRITES.DATA_WRITES.MISS

- **Title:** L3 Data Write Misses, **Category:** L3 Cache
- **Definition:** L3_WRITES.DATA_WRITES.MISS counts the number of L3 data write misses
- **Event Code:** 0x7E, **Umask:** 0110, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode matching: no, Data Address Range: no

LOADS_RETIRED

- **Title:** Retired Loads, **Category:** Memory
- **Definition:** LOADS_RETIRED counts the number of retired loads. The count includes integer, FP, RSE, VHPT, uncacheable loads and failed check loads (`ld.c`). Check loads that hit in the ALAT are *not* counted. Predicated-off operations are not counted
- **Event Code:** 0x6C, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: yes

MEMORY_CYCLE

- **Title:** Combined Memory Stall Cycles, **Category:** Stall
- **Definition:** MEMORY_CYCLE counts the number of cycles that the pipeline is stalled or flushed due to instructions waiting for data on cache misses, L1D way mispredictions, DTC misses, and RSE traffic.
- **Event Code:** 0x07, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

MISALIGNED_LOADS_RETIRED

- **Title:** Retired Unaligned Load Instructions, **Category:** Memory
- **Definition:** MISALIGNED_LOADS_RETIRED counts the number of retired unaligned loads that the hardware handled. The count includes integer, FP, and failed check loads (`ld.c`). Check loads that hit in the ALAT are *not* counted. Predicated-off operations are not counted
- **Event Code:** 0x70, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: yes

MISALIGNED_STORES_RETIRED

- **Title:** Retired Unaligned Store Instructions, **Category:** Memory
- **Definition:** MISALIGNED_STORES_RETIRED counts the number of retired unaligned store instructions that the hardware handled. The count includes integer, FP, and uncacheable stores. Predicated-off operations are not counted
- **Event Code:** 0x71, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: yes

NOPS_RETIRED

- **Title:** Retired Nop Instructions, **Category:** Execution
- **Definition:** NOPS_RETIRED counts the number of retired nop.i, nop.m or nop.b instructions. The count excludes predicated off nop instructions
- **Event Code:** 0x30, **Umask:** Ignored, **PMC/PMD:** 4, 5 **Max. Increment/Cycle:** 6
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

PIPELINE_ALL_FLUSH_CYCLE

- Title: Combination of Pipeline Flush Cycles caused by either a front-end or a back-end source,
  Category: Stall
- **Definition:** PIPELINE_ALL_FLUSH_CYCLE, for a given cycle, either counts the number of cycles spent during a front-end resteer of the pipeline (due to a correctly predicted taken branch), or counts the number of cycles spent during certain back-end resteers (due to a branch misprediction, ALAT flush or exception/serialization flush). This monitor does not count DTC flushes, way mispredictions, or floating-point flushes.
- **Event Code:** 0x04, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

PIPELINE_BACKEND_FLUSH_CYCLE

- **Title:** Combination of Pipeline Flush Cycles caused by either a Branch Misprediction or an Exception

  Category: Stall
- **Definition:** PIPELINE_BACKEND_FLUSH_CYCLE counts the number of cycles spent during back-end resteers of the pipeline (due to a branch misprediction, ALAT flush or exception/serialization flush). This monitor does not count DTC flushes, way mispredictions, or floating-point flushes.
- **Event Code:** 0x00, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1
- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

PIPELINE_FLUSH

- **Title:** Pipeline Flush, **Category:** System
- **Definition:** PIPELINE_FLUSH counts how often the Itanium processor pipeline is flushed due to IEU bypass conflict (caused by non-unit latency MMX operations such as variable shifts), data translation cache miss, L1 data cache way mispredict or other reasons such as an exception flush or an instruction serialization. Combinations of different flush reasons may be chosen by appropriately setting the umask. The monitor does not include branch misprediction flushes
- **Event Code:** 0x33, **Umask:** See below, **PMC/PMD:** 4, 5, 6, 7

Max. Increment/Cycle: 1

- **Qualification:** Instruction Address Range: no, Opcode Matching: no, Data Address Range: no

Table 7-32. Unit Mask Bits {19:18} for PIPELINE_FLUSH Event

FLUSH_TYPE	PMC.umask {19:16}	Description
IEU_FLUSH	1xxx	IEU bypass flush
DTC_FLUSH	x1xx	Data Translation Cache Miss flush
L1D_WAYMP_FLUSH	xx1x	L1 Way Misprediction flush
OTHER_FLUSH	xxx1	Other flush reason: exception flush or an instruction serialization.

PREDICATE_SQUASHED_RETIRED

- **Title:** Instructions Squashed Due to Predicate Off, **Category:** Execution
- **Definition:** PREDICATE_SQUASHED_RETIRED counts the number of instructions squashed due to a false qualifying predicate. The count includes all predicated off nops except nop.b's. Predicated off B-syllables (including nop.b) are not counted
- **Event Code:** 0x31, **Umask:** Ignored, **PMC/PMD:** 4, 5 **Max. Increment/Cycle:** 6
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: no

RSE_LOADS_RETIRED

- **Title:** RSE Load Accesses, **Category:** Execution
- **Definition:** RSE_LOADS_RETIRED counts the number of retired RSE loads
- **Event Code:** 0x72, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Refer to RSE_REFERENCES_RETIRED on page 7-61

RSE_REFERENCES_RETIRED

- **Title:** RSE Accesses, **Category:** Execution
- **Definition:** RSE_REFERENCES_RETIRED counts the number of retired RSE loads and stores
- **Event Code:** 0x65, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: yes

  RSE loads and stores are considered tagged if the `alloc`, `loadrs`, `flushrs` or branch return or `rfi` that caused the RSE references was tagged by the instruction address range or the opcode matcher. For data address range checking, the RSE reference is tagged only if its hits the programmed DBR range

STORES_RETIRED

- **Title:** Retired Stores, **Category:** Memory
- **Definition:** STORES_RETIRED counts the number of retired stores. The count includes integer, FP, RSE, and uncacheable stores. Predicated-off operations are not counted
- **Event Code:** 0x6D, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: yes

UC_LOADS_RETIRED

- **Title:** Retired Uncacheable Loads, **Category:** Memory
- **Definition:** UC_LOADS_RETIRED counts the number of retired uncacheable or write coalescing loads. The count includes integer, FP, RSE, and VHPT loads and failed check loads (ld.c). Check loads that hit in the ALAT are NOT counted. Predicated-off operations are not counted
- **Event Code:** 0x6E, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: yes

UC_STORES_RETIRED

- **Title:** Retired Uncacheable Stores, **Category:** Memory
- **Definition:** UC_STORES_RETIRED counts the number of retired uncacheable or write coalescing stores.The count includes integer, FP, RSE, and uncacheable stores. Predicated-off operations are not counted
- **Event Code:** 0x6F, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 2
- **Qualification:** Instruction Address Range: yes, Opcode Matching: yes, Data Address Range: yes

UNSTALLED_BACKEND_CYCLE

- **Title:** Unstalled Back-end Cycles**Category:** Stall
- **Definition:** UNSTALLED_BACKEND_CYCLE counts the number of cycles that the back-end is processing instructions without delay and the decoupling buffer between the front-end and back-end is empty, so that any effect on the front-end will be propagated to the back-end of the pipeline. This monitor thus reflects the number of cycles where there are no back-end stalls or flushes, and the decoupling buffer is empty, regardless of whether the L1I and ITLB are being hit or missed.
- **Event Code:** 0x05, **Umask:** Ignored, **PMC/PMD:** 4, 5, 6, 7 **Max. Increment/Cycle:** 1

Instruction Address Range: no, Opcode matching: no, Data Address Range: no