Intel® IA-64 Architecture Software Developer's Manual

Volume 4: Itanium™ Processor Programmer's Guide

January 2000

Order Number: 245320-001

THIS DOCUMENT IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Intel® IA-64 processors may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel’s website at http://developer.intel.com/design/litcentr.

Copyright © Intel Corporation, 2000

*Third-party brands and names are the property of their respective owners.

Contents

Figures

Tables

About this Manual 1

The IA-64 architecture is a unique combination of innovative features such as explicit parallelism, predication, speculation and more. The architecture is designed to be highly scalable to fill the ever increasing performance requirements of various server and workstation market segments. The IA-64 architecture features a revolutionary 64-bit instruction set architecture (ISA) which applies a new processor architecture technology called EPIC, or Explicitly Parallel Instruction Computing. A key feature of the IA-64 architecture is IA-32 instruction set compatibility.

The *Intel IA-64 Architecture Software Developer's Manual* provides a comprehensive description of the programming environment, resources, and instruction set visible to both the application and system programmer. In addition, it also describes how programmers can take advantage of IA-64 features to help them optimize code. This manual replaces the *IA-64 Application Developer's Architecture Guide* (Order Number 245188) which contains a subset of the information presented in this four-volume set.

1.1 Overview of *Volume 1: IA-64 Application Architecture*

This volume defines the IA-64 application architecture, including application level resources, programming environment, and the IA-32 application interface. This volume also describes optimization techniques used to generate high performance software.

1.1.1 Part 1: IA-64 Application Architecture Guide

Chapter 1, "About this Manual" provides an overview of all volumes in the *Intel IA-64 Architecture Software Developer's Manual*.

Chapter 2, "Introduction to the IA-64 Processor Architecture" provides an overview of the IA-64 architecture system environments.

Chapter 3, "IA-64 Execution Environment" describes the IA-64 register set used by applications and the memory organization models.

Chapter 4, "IA-64 Application Programming Model" gives an overview of the behavior of IA-64 application instructions (grouped into related functions).

Chapter 5, "IA-64 Floating-point Programming Model" describes the IA-64 floating-point architecture (including integer multiply).

Chapter 6, "IA-32 Application Execution Model in an IA-64 System Environment" describes the operation of IA-32 instructions within the IA-64 System Environment from the perspective of an application programmer.

1.1.2 Part 2: IA-64 Optimization Guide

Chapter 7, "About the IA-64 Optimization Guide" gives an overview of the IA-64 optimization guide.

Chapter 8, "Introduction to IA-64 Programming" provides an overview of the IA-64 application programming environment.

Chapter 9, "Memory Reference" discusses features and optimizations related to control and data speculation.

Chapter 10, "Predication, Control Flow, and Instruction Stream" describes optimization features related to predication, control flow, and branch hints.

Chapter 11, "Software Pipelining and Loop Support" provides a detailed discussion on optimizing loops through use of software pipelining.

Chapter 12, "Floating-point Applications" discusses current performance limitations in floating-point applications and IA-64 features that address these limitations.

1.2 Overview of *Volume 2: IA-64 System Architecture*

This volume defines the IA-64 system architecture, including system level resources and programming state, interrupt model, and processor firmware interface. This volume also provides a useful system programmer's guide for writing high performance system software.

1.2.1 Part 1: IA-64 System Architecture Guide

Chapter 1, "About this Manual" provides an overview of all volumes in the *Intel IA-64 Architecture Software Developer's Manual*.

Chapter 2, "IA-64 System Environment" introduces the environment designed to support execution of IA-64 operating systems running IA-32 or IA-64 applications.

Chapter 3, "IA-64 System State and Programming Model" describes the IA-64 architectural state which is visible only to an operating system.

Chapter 4, "IA-64 Addressing and Protection" defines the resources available to the operating system for virtual to physical address translation, virtual aliasing, physical addressing, and memory ordering.

Chapter 5, "IA-64 Interruptions" describes all interruptions that can be generated by an IA-64 processor.

Chapter 6, "IA-64 Register Stack Engine" describes the IA-64 architectural mechanism which automatically saves and restores the stacked subset (GR32 – GR 127) of the general register file.

Chapter 7, "IA-64 Debugging and Performance Monitoring" is an overview of the performance monitoring and debugging resources that are available in the IA-64 architecture.

Chapter 8, "IA-64 Interruption Vector Descriptions" lists all IA-64 interruption vectors.

Chapter 9, “IA-32 Interruption Vector Descriptions” lists IA-32 exceptions, interrupts and intercepts that can occur during IA-32 instruction set execution in the IA-64 System Environment.

Chapter 10, “IA-64 Operating System Interaction Model with IA-32 Applications” defines the operation of IA-32 instructions within the IA-64 System Environment from the perspective of an IA-64 operating system.

Chapter 11, “IA-64 Processor Abstraction Layer” describes the firmware layer which abstracts IA-64 processor implementation-dependent features.

1.2.2 Part 2: IA-64 System Programmer’s Guide

Chapter 12, “About the IA-64 System Programmer’s Guide” gives an introduction to the second section of the system architecture guide.

Chapter 13, “MP Coherence and Synchronization” describes IA-64 multi-processing synchronization primitives and the IA-64 memory ordering model.

Chapter 14, “Interruptions and Serialization” describes how the processor serializes execution around interruptions and what state is preserved and made available to low-level system code when interruptions are taken.

Chapter 15, “Context Management” describes how operating systems need to preserve IA-64 register contents and state. This chapter also describes IA-64 system architecture mechanisms that allow an operating system to reduce the number of registers that need to be spilled/filled on interruptions, system calls, and context switches.

Chapter 16, “Memory Management” introduces various IA-64 memory management strategies.

Chapter 17, “Runtime Support for Control and Data Speculation” describes the operating system support that is required for control and data speculation.

Chapter 18, “Instruction Emulation and Other Fault Handlers” describes a variety of instruction emulation handlers that IA-64 operating system are expected to support.

Chapter 19, “Floating-point System Software” discusses how IA-64 processors handle floating-point numeric exceptions and how the IA-64 software stack provides complete IEEE-754 compliance.

Chapter 20, “IA-32 Application Support” describes the support an IA-64 operating system needs to provide to host IA-32 applications.

Chapter 21, “External Interrupt Architecture” describes the IA-64 external interrupt architecture with a focus on how external asynchronous interrupt handling can be controlled by software.

Chapter 22, “I/O Architecture” describes the IA-64 I/O architecture with a focus on platform issues and support for the existing IA-32 I/O port space platform infrastructure.

Chapter 23, “Performance Monitoring Support” describes the IA-64 performance monitor architecture with a focus on what kind of operating system support is needed from IA-64 operating systems.

Chapter 24, “Firmware Overview” introduces the IA-64 firmware model, and how various firmware layers (PAL, SAL, EFI) work together to enable processor and system initialization, and operating system boot.

1.2.3 Appendices

Appendix A, “IA-64 Resource and Dependency Semantics” summarizes the dependency rules that are applicable when generating code for IA-64 processors.

Appendix B, “Code Examples” provides OS boot flow sample code.

1.3 Overview of *Volume 3: Instruction Set Reference*

This volume is a comprehensive reference to the IA-64 and IA-32 instruction sets, including instruction format/encoding.

1.3.1 Part 1: IA-64 Instruction Set Descriptions

Chapter 1, “About this Manual” provides an overview of all volumes in the *Intel IA-64 Architecture Software Developer’s Manual*.

Chapter 2, “IA-64 Instruction Reference” provides a detailed description of all IA-64 instructions, organized in alphabetical order by assembly language mnemonic.

Chapter 3, “IA-64 Pseudo-Code Functions” provides a table of pseudo-code functions which are used to define the behavior of the IA-64 instructions.

Chapter 4, “IA-64 Instruction Formats” describes the encoding and instruction format instructions.

1.3.2 Part 2: IA-32 Instruction Set Descriptions

Chapter 5, “Base IA-32 Instruction Reference” provides a detailed description of all base IA-32 instructions, organized in alphabetical order by assembly language mnemonic.

Chapter 6, “IA-32 MMX™ Technology Instruction Reference” provides a detailed description of all IA-32 MMX™ technology instructions designed to increase performance of multimedia intensive applications. Organized in alphabetical order by assembly language mnemonic.

Chapter 7, “IA-32 Streaming SIMD Extension Instruction Reference” provides a detailed description of all IA-32 Streaming SIMD Extension instructions designed to increase performance of multimedia intensive applications, and is organized in alphabetical order by assembly language mnemonic.

1.4 Overview of Volume 4: *Itanium™ Processor Programmer's Guide*

This volume describes model-specific architectural features incorporated into the Intel® Itanium™ processor, the first IA-64 processor.

Chapter 1, "About this Manual" provides an overview of four volumes in the *Intel IA-64 Architecture Software Developer's Manual.*

Chapter 2, "Register Stack Engine Support" summarizes Register Stack Engine (RSE) support provided by the Itanium processor.

Chapter 3, "Virtual Memory Management Support" details size of physical and virtual address, region register ID, and protection key register implemented on the Itanium processor.

Chapter 4, "Processor Specific Write Coalescing (WC) Behavior" describes the behavior of write coalesce (also known as Write Combine) on the Itanium processor.

Chapter 5, "Model Specific Instruction Implementation" describes model specific behavior of IA-64 instructions on the Itanium processor.

Chapter 6, "Processor Performance Monitoring" defines the performance monitoring features which are specific to the Itanium processor. This chapter outlines the targeted performance monitor usage models and describes the Itanium processor specific performance monitoring state.

Chapter 7, "Performance Monitor Events" summarizes the Itanium processor events and describes how to compute commonly used performance metrics for Itanium processor events.

Chapter 8, "Model Specific Behavior for IA-32 Instruction Execution" describes some of the key differences between an Itanium processor executing IA-32 instructions and the Pentium III processor.

1.5 Terminology

The following definitions are for terms related to the IA-64 architecture and will be used throughout this document:

Instruction Set Architecture (ISA) – Defines application and system level resources. These resources include instructions and registers.

IA-64 Architecture – The new ISA with 64-bit instruction capabilities, new performance-enhancing features, and support for the IA-32 instruction set.

IA-32 Architecture – The 32-bit and 16-bit Intel Architecture as described in the *Intel Architecture Software Developer's Manual*.

IA-64 Processor – An Intel 64-bit processor that implements both the IA-64 and the IA-32 instruction sets.

IA-64 System Environment – The IA-64 operating system privileged environment that supports the execution of both IA-64 and IA-32 code.

IA-32 System Environment – The operating system privileged environment and resources as defined by the *Intel Architecture Software Developer's Manual*. Resources include virtual paging, control registers, debugging, performance monitoring, machine checks, and the set of privileged instructions.

IA-64 Firmware – The Processor Abstraction Layer (PAL) and System Abstraction Layer (SAL).

Processor Abstraction Layer (PAL) – The IA-64 firmware layer which abstracts IA-64 processor features that are implementation dependent.

System Abstraction Layer (SAL) – The IA-64 firmware layer which abstracts IA-64 system features that are implementation dependent.

1.6 Related Documents

The following documents contain additional material related to the *Intel® IA-64 Architecture Software Developer's Manual*:

- ***Intel Architecture Software Developer's Manual*** – This set of manuals describes the Intel 32-bit architecture. They are readily available from the Intel Literature Department by calling 1-800-548-4725 and requesting Order Numbers 243190, 243191and 243192, or can be downloaded at http://developer.intel.com/design/litcentr.
- ***IA-64 Software Conventions and Runtime Architecture Guide* –** This document defines general information necessary to compile, link, and execute a program on an IA-64 operating system. It can be downloaded at http://developer.intel.com/design/ia64.
- ***IA-64 System Abstraction Layer Specification*** – This document specifies requirements to develop platform firmware for IA-64 processor systems.
- ***Extensible Firmware Interface Specification*** – This document defines a new model for the interface between operating systems and platform firmware. It can be downloaded at http://developer.intel.com/technology/efi.

Register Stack Engine Support 2

2.1 RSE Modes

The Itanium processor implements the enforced lazy RSE mode. Refer to Chapter 6, “IA-64 Register Stack Engine” in Volume 2 for a description of the RSE modes.

2.2 RSE and Clean Register Stack Partitions

On the Itanium processor, the internal RSE pointer RSE.BSPLoad is always equal to AR.BSPStore, meaning that the size of the clean register stack partition is always zero. This implies that, on the Itanium processor, a `flushrs` instruction will create a dirty region of size zero and an invalid region of size equal to 96 - CFM.sof. On other implementations that maintain a clean partition, `flushrs` behavior may differ by creating a clean register stack partition in addition to an invalid partition and a zero-sized dirty partition. As a result, the Itanium processor’s RSE may perform more mandatory fills upon a branch-return (`br.ret`) or `rfi` following a `flushrs` instruction than an implementation that maintains a clean partition.

Virtual Memory Management Support 3

3.1 Page Size Supported

The following page sizes are supported on the Itanium processor: 4K, 8K, 16K, 64K, 256K, 1M, 4M, 16M and 256M bytes.

3.2 Physical and Virtual Addresses

The IA-64 architecture requires that a processor implement at least 54 virtual address bits and 32 physical address bits. The Itanium processor implements 54 virtual address bits (51 address bits plus 3 region index bits) and 44 physical address bits.

3.3 Region Register ID

The Itanium processor implements the minimum region register IDs allowed by the IA-64 architecture. The region register ID contains 18 bits.

3.4 Protection Key Register

The IA-64 architecture requires a minimum of 16 protection key registers, each at least as wide as the region register IDs. The Itanium processor implements 16 protection key registers, each 21 bits wide.

Processor Specific Write Coalescing (WC) Behavior

4

4.1 Write Coalescing

For increased performance of uncacheable references to frame buffers, previous Intel IA-32 processors defined the Write Coalescing (WC) memory type. WC coalesces streams of data writes into a single larger bus write transaction. Refer to the *Intel Architecture Software Developer's Manual* for additional information.

On the Itanium processor, WC loads are performed directly from memory and not from coalescing buffers. It has a separate 2-entry, 64-byte Write Coalesce Buffer (WCB) which is used exclusively for WC accesses. Each byte in the line has a valid bit. If all the valid bits are true, then the line is full and will be evicted (or flushed) by the processor.

Note: WC behavior of the Itanium processor in the IA-32 System Environment is similar to the Pentium III processor. Refer to the *Intel Architecture Software Developer's Manual* for more information.

4.2 WC Buffer Eviction Conditions

To ensure consistency with memory, the WCB is flushed on the following conditions (both entries are flushed). These conditions are followed when the processor is operating in the IA-64 System Environment:

Table 4-1. Itanium™ Processor WCB Eviction Conditions

Eviction Condition	IA-64 Instructions
Memory Fence (mf)	mf
Architectural Conditions for WCB Flush	
Memory Release ordering (op.rel)	st.rel, cmpxchg.rel, fetchadd.rel, ptc.g
Flush Cache (fc) hit on WCB	yes
Flush Write Buffers (fwb)	yes
Any UC load	no [a]
Any UC store	no [a]
UC load or ifetch hits WCB	no [a]
UC store hits WCB	no [a]
WC load/ifetch hits WCB	
WC store hits WCB	

a. IA-64 architecture doesn't require the WC buffers to be coherent w.r.t to UC load/store operations.

4.3 WC Buffer Flushing Behavior

As mentioned previously, the Itanium processor WCB contains two entries. The WC entries are flushed in the same order as they are allocated. That is, the entries are flushed in written order. This flushing order applies only to a “well-behaved” stream. A “well-behaved” stream writes one WC entry at a time and does not write the second WC entry until the first one is full.

In the absence of platform retry or deferral, the flushing rule implies that the WCB entries are always flushed in a program written order for a “well-behaved” stream, even in the presence of interrupts. For example, consider the following scenario: if software issues a “well-behaved” stream, but is interrupted in the middle; one of the WC entries could be partially filled. The WCB (including the partially filled entry) could be flushed by the OS kernel code or by other processes. When the interrupted context resumes, it sends out the remaining line and then moves on to fill the other entry. Note that the resumed context could be interrupted again in the middle of filling up the other entry, causing both entries to be partially filled when the interrupt occurs.

For streams that do not conform to the above “well-behaved” rule, the order in which the WC buffer is flushed is random.

WCB eviction is performed for full lines by a single 64-byte bus transaction in a stream of 8-byte packages. For partially full lines, the WCB is evicted using up to eight 8-byte transactions with the proper byte enables. When flushing, WC transactions are given the highest priority of all external bus operations.

Model Specific Instruction Implementation

5

This section describes how IA-64 instructions with processor implementation-specific features, behave on the Intel Itanium processor.

5.1 ld.bias

If the instruction hits L1D[1] or L2 cache and the state of the line is exclusive (E) or modified (M), the line is returned and remains in cache; no external bus traffic is generated. If the line is shared (S) or invalid (I) or the instruction misses the L2, it is treated as a store miss by the L3/bus. The line is returned and stored in E state by the processor in the L2 and L3 cache.

Please refer to page 2-126 in Volume 3 for a detailed description of the `ld` instruction.

5.2 lfetch Exclusive Hint

The exclusive hint in the `lfetch` instruction allows the cache line to be fetched in an exclusive (E) state. On the Itanium processor, an `lfetch` transaction that has a snoop hit will be cached in an shared (S) state; otherwise, it is cached in an exclusive state.

Please refer to page 2-137 in Volume 3 for a detailed description of the `lfetch` instruction.

5.3 fwb

The Itanium processor implements the flush write-back buffer (`fwb`) instruction. This instruction carries a weak memory attribute and causes the coalescing buffer to be flushed. The L1D and L2 store buffers are not flushed.

Please refer to page 2-117 in Volume 3 for a detailed description of the `fwb` instruction.

---

[1] The Intel Itanium™ processor cache hierarchy consists of the following levels: on-chip L1I, L1D, L2 caches, and off-chip L3 cache.

5.4 thash

The IA-64 architecture defines a `thash` instruction for generating the hash address for long format VHPT. `thash` is implementation specific. On the Itanium processor, since the hashing function is performed in the HPW, the HPW will generate the VHPT Entry which corresponds to the virtual address supplied. The hashing function is given in the following pseudo-code:

```
If (GR[r3].nat = ’1 or unimplemented virtual address bits) then {
    GR[r1] = ’0 ;                           // treated as a speculative access.
    GR[r1].nat = ’1;
    }
else {
    Mask = (2^PTA.size) - 1;
    HPN = VA{50:0} >> RR[VA{63:61}].ps;     // Hash Page Number unsigned right shift.
                                            // mov 2 RR checks for supported ps
    if (PTA.vf=32) {                        // 32B PTE (Long format)
        Hash_Index = HPN ^ (zero{63:18} || rid{17:0})
        VHPT_Offset = Hash_Index << 5 ;
    }
    if (PTA.vf=8) {                         // 8B PTE
        Hash_Index = HPN ;
        VHPT_Offset = Hash_Index << 3;
    }
    GR[r1] =  (PTA.base{63:61} << 61)
                || ([(PTA.base{60:15} & ~Mask{60:15}) ||
                        (VHPT_Offset{60:15} & Mask{60:15})] << 15)
                || VHPT_Offset{14:0} ;
    }
}
```

Please refer to page 2-224 in Volume 3 for a detailed description of the `thash` instruction.

5.5 ttag

The IA-64 architecture defines the `ttag` instruction for generating the tag for a long format VHPT entry. `ttag` is implementation specific. The HPW will generate the tag for the long format VHPT entry which corresponds to the virtual address supplied. The function is:

```
If (GR[r3].nat = ’1 or unimplemented virtual address bits) then {
    GR[r1] = ’0 ;
    GR[r1].nat = ’1;
    }
else {
    GR[r1] =(VA{50:0}>> RR[VA{63:61}].PS) ^
                ((zero{5:0} || RR[VA{63:61}].RID{17:0}) << 39);
    }
}
```

Please refer to page 2-228 in Volume 3 for a detailed description of the `ttag` instruction.

5.6 ptc.e

On the Itanium processor, a single `ptc.e` purges all translation cache (TC) entries in both the instruction and data TLBs. The caches are not flushed.

Please refer to page 2-192 in Volume 3 for a detailed description of the `ptc` instruction.

5.7 mf.a

In the IA-64 architecture, the `mf.a` instruction is a memory acceptance fence for UC transactions only. On the Itanium processor, `mf.a` is implemented as an acceptance fence for both cacheable and UC data transactions (but not I fetches). The processor stalls until all data buffers in the L2 and bus are empty. This does not include buffers for instruction and L3 WB buffer in the bus request queue.

Please refer to page 2-140 in Volume 3 for a detailed description of the `mf` instruction.

5.8 Prefetch Behavior

The Itanium processor does not initiate prefetches with post-increment loads.

5.9 Temporal and Non-temporal Hints Support

IA-64 architecture provides memory locality hints for data accesses that can be used for allocation control in the processor cache hierarchy. For more details on this topic, please refer to Volume 1, Section 4.4.6. Implementation of locality hints is left as an implementation-specific feature on IA-64 processors.

On the Itanium processor, four types of memory locality hints are implemented: t1, nt1, nt2 and nta. The Itanium processor does not support a non-temporal buffer; instead, non-temporal accesses are allocated in L2 cache with biased replacement.

Processor Performance Monitoring 6

This chapter defines the performance monitoring features on the Itanium processor. The Itanium processor provides four 32-bit performance counters, more than 50 monitorable events, and several advanced monitoring capabilities. This chapter outlines the targeted performance monitor usage models, defines the software interface and programming model, and lists the set of monitored events.

IA-64 architecture incorporates architected mechanisms that allow software to actively and directly manage performance critical processor resources such as branch prediction structures, processor data and instruction caches, virtual memory translation structures, and more. To achieve the highest performance levels, dynamic processor behavior can be monitored and fed back into the code generation process to improve observed run-time behavior or to expose higher levels of instruction level parallelism. One can quantify and measure behavior of real-world IA-64 applications, tools and operating systems. These measurements will be critical for compiler optimizations and the efficient use of several architectural features such as speculation, predication, and more.

The remainder of this chapter is split into the following two subsections:

- Section 6.1, "Performance Monitor Programming Models" discusses how performance monitors are used and presents various Itanium processor performance monitoring programming models.
- Section 6.2, "Performance Monitor State" defines the Itanium processor specific PMC/PMD performance monitoring registers.

6.1 Performance Monitor Programming Models

This section introduces the Itanium processor performance monitoring features from a programming model point-of-view and describes how the different event monitoring mechanisms can be used effectively. The Itanium processor performance monitor architecture focuses on the following two usage models:

- **Workload Characterization**: the first step in any performance analysis is to understand the performance characteristics of the workload under study. Section 6.1.1, "Workload Characterization" discusses the Itanium processor support for workload characterization.
- **Profiling**: profiling is used by application developers and profile-guided compilers. Application developers are interested in identifying performance bottlenecks and relating them back to their code. Their primary objective is to understand which program location caused performance degradation at the module, function, and basic block level. For optimization of data placement and the analysis of critical loops, instruction level granularity is desirable. Profile-guided compilers that use advanced IA-64 architectural features such as predication and speculation benefit from run-time profile information to optimize instruction schedules. The Itanium processor supports instruction granular statistical profiling of branch mispredicts and cache misses. Details of the Itanium processor's profiling support are described in Section 6.1.2, "Profiling".

Whenever monitoring overhead is irrelevant, but accuracy is the primary objective, system and processor designers may resort to tracing processor activity at the system or the processor bus interface. However, trace based performance analysis and hardware tracing of the Itanium processor are beyond the scope of this documentation.

6.1.1 Workload Characterization

The first step in any performance analysis is to understand the performance characteristics of the workload under study. There are two fundamental measures of interest: event rates and program cycle break down.

- **Event Rate Monitoring**: Event rates of interest include average retired instructions-per-clock (IPC), data and instruction cache miss rates, or branch mispredict rates measured across the entire application. Characterization of operating systems or large commercial workloads (e.g. OLTP analysis) requires a system-level view of performance relevant events such as TLB miss rates, VHPT walks/second, interrupts/second or bus utilization rates. Section 6.1.1.1, "Event Rate Monitoring" discusses event rate monitoring.
- **Cycle Accounting**: The cycle break-down of a workload attributes a reason to every cycle spent by a program. Apart from a program's inherent execution latency, extra cycles are usually due to pipeline stalls and flushes. Section 6.1.1.4, "Cycle Accounting" discusses cycle accounting.

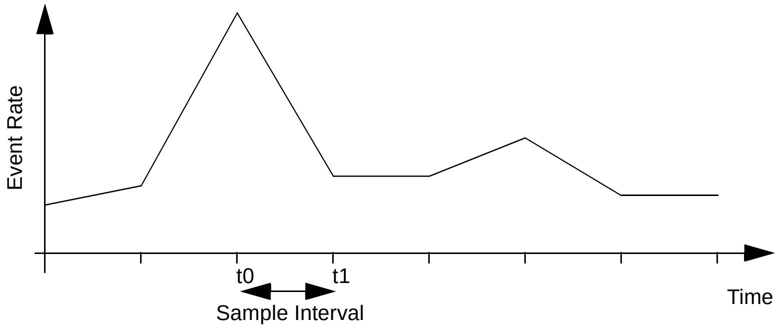
6.1.1.1 Event Rate Monitoring

Event rate monitoring determines event rates by reading processor event occurrence counters before and after the workload is run and then computing the desired rates. For instance, two basic Itanium processor events that count the number of retired IA-64 instructions (IA64_INST_RETIRED.u) and the number of elapsed clock cycles (CPU_CYCLES) allow a workload's instructions per cycle (IPC) to be computed as follows:

$$\text{IPC} = (\text{IA64\_INST\_RETIRED.u}_{t1} - \text{IA64\_INST\_RETIRED.u}_{t0}) / (\text{CPU\_CYCLES}_{t1} - \text{CPU\_CYCLES}_{t0})$$

Time-based sampling is the basis for many performance debugging tools [VTune, gprof, Windows NT*]. As shown in Figure 6-1, time-based sampling can be used to plot the event rates over time, and can provide insights into the different phases the workload moves through.

Figure 6-1. Time-based Sampling



On the Itanium processor, many event types (e.g. TLB misses or branch mispredicts) are limited to a rate of one per clock cycle. These are referred to as “single occurrence” events. However, in the Itanium processor multiple events of the same type may occur in the same clock. We refer to such events as “multi-occurrence” events. An example of a multi-occurrence events on the Itanium processor is data cache misses (up to two per clock). Multi-occurrence events, such as the number of entries in the memory request queue, can be used to the derive average number and average latency of memory accesses. The next two sections describe the basic Itanium processor mechanisms for monitoring single and multi-occurrence events.

6.1.1.2 Single Occurrence Events and Duration Counts

A single occurrence event can be monitored by any of the Itanium processor performance counters. For all single occurrence events a counter is incremented by up to one per clock cycle. Duration counters that count the number of clock cycles during which a condition persists are considered “single occurrence” events. Examples of single occurrence events on the Itanium processor are TLB misses, branch mispredictions, or cycle-based metrics.

6.1.1.3 Multi-occurrence Events, Thresholding and Averaging

Events that, due to hardware parallelism, may occur at rates greater than one per clock cycle are termed “multi-occurrence” events. Examples of such events on the Itanium processor are retired instructions or the number of live entries in the memory request queue. The Itanium processor’s four performance counters are asymmetrical. While all counters handle single-occurrence and multi-occurrence events with event rates up to three per cycle, only two counters can handle multi-occurrence events with event rates up to seven per cycle. For details, see Section 6.2.2, "Performance Counter Registers".

Thresholding capabilities are available in the Itanium processor’s multi-occurrence counters and can be used to plot an event distribution histogram. When a non-zero threshold is specified, the monitor is incremented by one in every cycle in which the observed event count exceeds that programmed threshold. This allows questions such as “for how many cycles did the memory request queue contain more than two entries?” or “during how many cycles did the machine retire more than three instructions?” to be answered. This capability allows micro-architectural buffer sizing experiments to be supported by real measurements. By running a benchmark with different threshold values, a histogram can be drawn up that may help to identify the performance “knee” at a certain buffer size.

For overlapping concurrent events, such as pending memory operations, the average number of concurrently outstanding requests and the average number of cycles that requests were pending is of interest. To calculate the average number or latency of multiple outstanding requests in the memory queue, we need to know the total number of requests (n_{total}) and, in each cycle, the number of live requests per cycle (n_{live}/cycle). By summing up the live requests (n_{live}/cycle) using a multi-occurrence counter Σnlive is directly measured by hardware. We can now calculate the average number of requests and the average latency as follows:

- Average outstanding requests/cycle = $\Sigma n_{live} / \Delta t$
- Average latency per request = $\Sigma n_{live} / n_{total}$

An example of this calculation is given in Table 6-1, in which the average outstanding requests/cycle = 15/8 = 1.825, and the average latency per request = 15/5 = 3 cycles.

Table 6-1. Average Latency per Request and Requests per Cycle Calculation Example

Time [Cycles]	1	2	3	4	5	6	7	8
# Requests In	1	1	1	1	1	0	0	0
# Requests Out	0	0	0	1	1	1	1	1
n_{live}	1	2	3	3	3	2	1	0
Σn_{live}	1	3	6	9	12	14	15	15
n_{total}	1	2	3	4	5	5	5	5

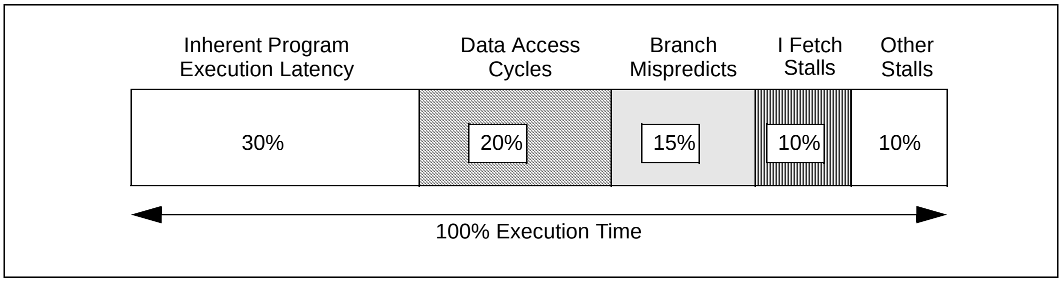
The Itanium processor provides the following capabilities to support event rate monitoring:

- Clock cycle counter
- Retired instruction counter
- Event occurrence and duration counters
- Multi-occurrence counters with thresholding capability

6.1.1.4 Cycle Accounting

While event rate monitoring counts the number of events, it does not tell us whether the observed events are contributing to a performance problem. A commonly used strategy is to plot multiple event rates and correlate them with the measured instructions per cycle (IPC) rate. If a low IPC occurs concurrently with a peak of cache miss activity, chances are that cache misses are causing a performance problem. To eliminate such guess work, the Itanium processor provides a set of IA-64 cycle accounting monitors, that break-down the number of cycles that are lost due to various kinds of micro-architectural events. As shown in Figure 6-2, this lets us account for every cycle spent by a program and therefore provides insight into an application's micro-architectural behavior. Note that cycle accounting is different from simple stall or flush duration counting. Cycle accounting is based on the machine's actual stall and flush conditions and accounts for overlapped pipeline delays, while simple stall or flush duration counters do not. Cycle accounting determines a program's cycle break-down by stall and flush reasons, while simple duration counters are useful in determining cumulative stall or flush latencies.

Figure 6-2. IA-64 Cycle Accounting



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. Branch Mispredict Cycle: branch mispredicts, pipeline flushes (includes interrupts and exceptions)
2. Data Access Cycle: memory pipeline full, data TLB stalls, and load-use stalls
3. Execution Latency Cycle: scoreboard stalls and FPU stalls
4. RSE Active Cycle: RSE spill/fill stall
5. Issue Limit Cycle: instruction issue, stops, or resource oversubscription stalls
6. Instruction Access Cycle: instruction fetch stalls due to instruction cache or TLB misses
7. Taken Branch Cycle: instruction fetch branch bubbles
8. Fetch Window Cycle: partial instruction fetch stalls due to non instruction cache line aligned branch targets

Four of the eight categories (1,2,3,6) are directly measurable as the 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: branch cycles (1+7), memory cycles (2+4), execution cycles (3+5), and instruction fetch cycles (6+8) are directly measurable as a Itanium processor event. For details refer to Section 7.3, “Cycle Accounting Events” on page 7-5.

6.1.2 Profiling

Profiling is used by application developers and profile-guided compilers, optimizing linkers and run-time systems. Application developers are interested in identifying performance bottlenecks and relating them back to their source code. Based on profile feedback developers can make changes to the high-level algorithms and data structures of the program. Compilers can use profile feedback to optimize instruction schedules by employing advanced IA-64 architectural features such as predication and speculation.

To support profiling, performance monitor counts have to be associated with program locations. The following mechanisms are supported directly by the Itanium processor’s performance monitors:

- Program Counter Sampling
- Miss Event Address Sampling: Itanium processor Event Address Registers (EARs) provide sub-pipeline length event resolution for performance critical events (instruction and data caches, branch mispredicts, instruction and data TLBs).
- Event Qualification: constrains event monitoring to a specific instruction address range, to certain opcodes or privilege levels.

These profiling features are presented in the next three subsections.

6.1.2.1 Program Counter Sampling

Application tuning tools like [VTune, gprof] use time-based or event-based sampling of the program counter and other event counters to identify performance critical functions and basic blocks. As shown in Figure 6-3, the sampled points can be histogrammed by instruction addresses. For application tuning, statistical sampling techniques have been very successful, because the programmer can rapidly identify code hot-spots in which the program spends a significant fraction of its time or where certain event counts are high.

Figure 6-3. Event Histogram by Program Counter

Program counter sampling points the performance analysts at code hot-spots, but does not indicate what caused the performance problem. Inspection and manual analysis of the hot-spot region along with a fair amount of guess work are required to identify the root cause of the performance problem. On the Itanium processor, the cycle accounting mechanism (described in Section 6.1.1.4, "Cycle Accounting") can be used to directly measure an application's micro-architectural behavior.

The IA-64 architectural interval timer facilities (ITC and ITM registers) can be used for time-based program counter sampling. Event-based program counter sampling is supported by a dedicated performance monitor overflow interrupt mechanism described in detail in Volume 2, Section 7.2.2, "Performance Monitor Overflow Status Registers (PMC[0]..PMC[3])".

To support program counter sampling, the Itanium processor provides the following mechanisms:

- Timer interrupt for time-based program counter sampling.
- Event count overflow interrupt for event-based program counter sampling.
- Hardware supported cycle accounting.

6.1.2.2 Miss Event Address Sampling

Program counter sampling and cycle accounting provide an accurate picture of cumulative micro-architectural behavior, but they do not provide the application developer with pointers to specific program elements (code locations and data structures) that repeatedly cause micro-architectural "miss events". In a cache study of the SPEC92 benchmarks, [Lebeck] used (trace based) cache miss profiling to gain performance improvements of 1.02 to 3.46 on various benchmarks by making simple changes to the source code. This type of analysis requires identification of instruction and data addresses related to micro-architectural "miss events" such as cache misses, branch mispredicts, or TLB misses. Using symbol tables or compiler annotations these addresses can be mapped back to critical source code elements. Like Lebeck, most performance analysts in the past have had to capture hardware traces and resort to trace driven simulation.

Due to the super-scalar issue, deep pipelining, and out-of-order instruction completion of today's micro-architectures, the sampled program counter value may not be related to the instruction address that caused a miss event. On a Pentium processor pipeline, the sampled program counter may be off by 2 dynamic instructions from the instruction that caused the miss event. On a Pentium Pro processor, this distance increases to approximately 32 dynamic instructions. On the Itanium processor it is approximately 48 dynamic instructions. If program counter sampling is used for miss event address identification on the Itanium processor, a miss event might be associated with an instruction almost five dynamic basic blocks away from where it actually occurred (assuming that 10% of all instructions are branches). Therefore, it is essential for hardware to precisely identify an event's address.

The Itanium processor provides a set of *event address registers* (EARs) that record the instruction and data addresses of data cache misses for loads, the instruction and data addresses of data TLB misses, the instruction addresses of instruction TLB and cache misses. A four deep *branch trace buffer* captures sequences of branch instructions. Table 6-2 summarizes the capabilities offered by the EARs and branch trace buffer. Exposing miss event addresses to software allows them to be monitored either by sampling or by code instrumentation. This eliminates the need for trace generation to identify and solve performance problems and enables performance analysis by a much larger audience on unmodified hardware.

Table 6-2. Itanium™ Processor EARs and Branch Trace Buffer

Event Address Register	Triggers on	What is Recorded
Instruction Cache	Instruction fetches that miss the L1 instruction cache (demand fetches only)	Instruction Address Number of cycles fetch was in flight
Instruction TLB (ITLB)	Instruction fetch missed ITLB (demand fetches only)	Instruction Address Who serviced TLB miss: VHPT or software
Data Cache	Load instructions that miss L1 data cache	Instruction Address Data Address Number of cycles load was in flight.
Data TLB (DTLB)	Data references that miss L1 DTLB	Instruction Address Data Address Who serviced TLB miss: L2 DTLB, VHPT or software
Branch Trace Buffer	Branch Outcomes	Branch Instruction Address Branch Target Instruction Address Mispredict status and reason

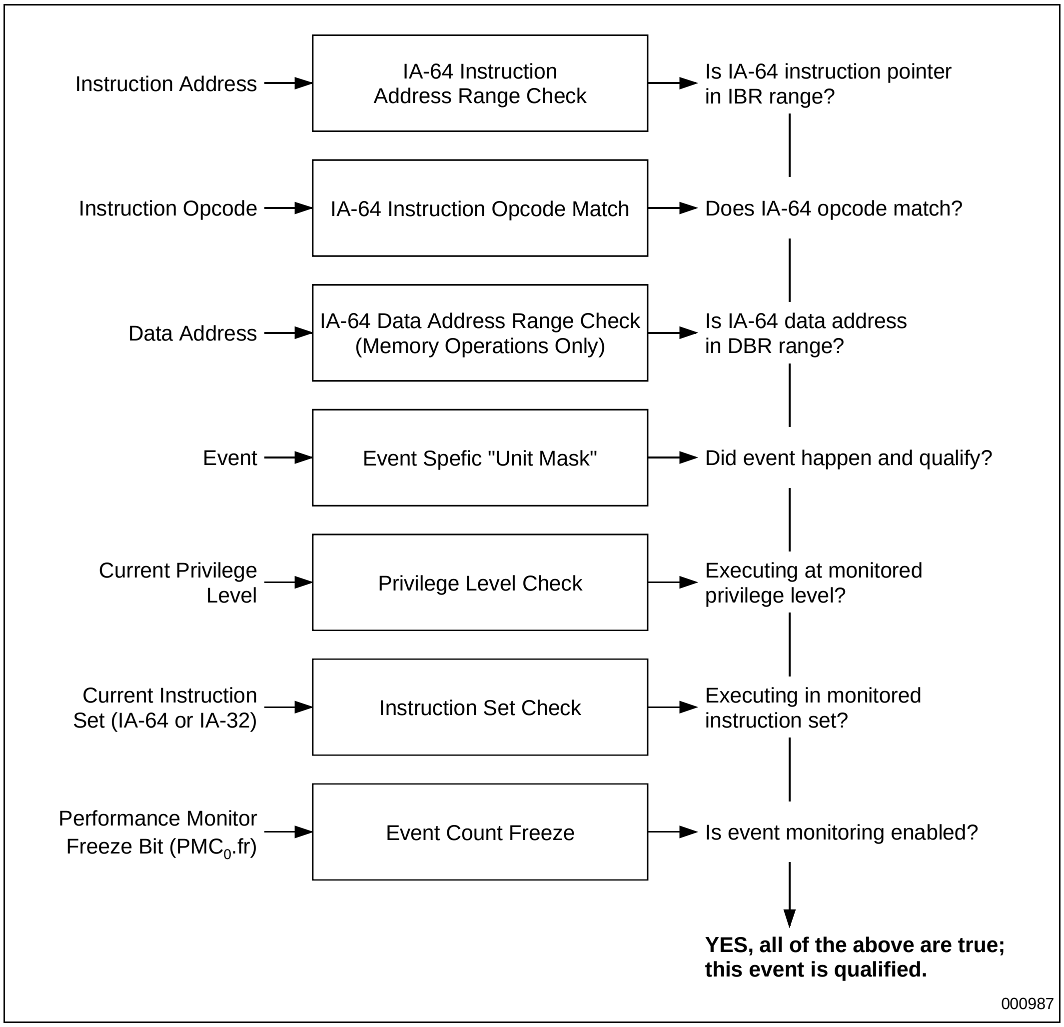
The Itanium processor EARs enable statistical sampling by configuring a performance counter to count, for instance, the number of data cache misses or retired instructions. The performance counter value is set up to interrupt the processor after a pre-determined number of events have been observed. The data cache event address register repeatedly captures the instruction and data addresses of actual data cache load misses. Whenever the counter overflows, miss event address collection is suspended until the event address register is read by software (this prevents software from capturing a miss event that might be caused by the monitoring software itself). When the counter overflows an interrupt is delivered to software, the observed event addresses are collected, and a new observation interval can be setup by rewriting the performance counter register. For time-based (rather than event-based) sampling methods, the event address registers indicate to software whether or not a qualified event was captured. Statistical sampling can achieve arbitrary event resolution by varying the number of events within an observation interval, and by increasing the number of observation intervals.

6.1.3 Event Qualification

On the Itanium processor, performance monitoring can be confined to a subset of all events. As shown in Figure 6-4, events can be qualified for monitoring based on an instruction address range, a particular instruction opcode, a data address range, an event specific "unit-mask", the privilege level and instruction set the event was caused by, and the status of the performance monitoring freeze bit (PMC[0].fr).

- IA-64 Instruction Address Range Check: The Itanium processor allows event monitoring to be constrained to a programmable instruction address range. This enables monitoring of dynamically linked libraries (DLL), functions, or loops of interest in the context of a large IA-64 application. The IA-64 instruction address range check is applied at the instruction fetch stage of the pipeline and the resulting qualification is carried by the instruction throughout the pipeline. This enables conditional event counting at a level of granularity smaller than dynamic instruction length of the pipeline (approximately 48 instructions). The Itanium processor's instruction address range check operates only during IA-64 code execution (i.e. when PSR.is is zero). For details, see Section 6.2.4, "IA-64 Instruction Address Range Check Register (PMC[13])".
- IA-64 Instruction Opcode Match: The Itanium processor provides two independent IA-64 opcode match registers each of which match the currently issued instruction encodings with a programmable opcode match and mask function. The resulting match events can be selected as an event type for counting by the performance counters. This allows histogramming of instruction types, usage of destination and predicate registers as well as basic block profiling (through insertion of tagged `nops`). The opcode matcher operates only during IA-64 code execution (i.e. when PSR.is is zero). Details are described in Section 6.2.5, "IA-64 Opcode Match Registers (PMC[8,9])".
- IA-64 Data Address Range Check: The Itanium processor allows event collection for memory operations to be constrained to a programmable data address range. This enables selective monitoring of data cache miss behavior of specific data structures. For details, see Section 6.2.6, "IA-64 Data Address Range Check (PMC[11])".
- Event Specific Unit Masks: Some events allow the specification of "unit masks" to filter out interesting events directly at the monitored unit. For details, refer to the event pages in Chapter 7, "Performance Monitor Events".
- Privilege Level: Two bits in the processor status register are provided to enable selective process-based event monitoring. The Itanium processor supports conditional event counting based on the current privilege level; this allows performance monitoring software to break-down event counts into user and operating system contributions. For details on how to constrain monitoring by privilege level refer to Section 6.2.1, "Performance Monitor Control and Accessibility".
- Instruction Set: The Itanium processor supports conditional event counting based on the currently executing instruction set (IA-64 or IA-32) by providing two instruction set mask bits for each event monitor. This allows performance monitoring software to break-down event counts into IA-64 and IA-32 contributions. For details, refer to Section 6.2.1, "Performance Monitor Control and Accessibility".
- Performance Monitor Freeze: Event counter overflows or software can freeze event monitoring. When frozen, no event monitoring takes place until software clears the monitoring freeze bit (PMC[0].fr). This ensures that the performance monitoring routines themselves, e.g. counter overflow interrupt handlers or performance monitoring context switch routines, do not "pollute" the event counts of the system under observation.

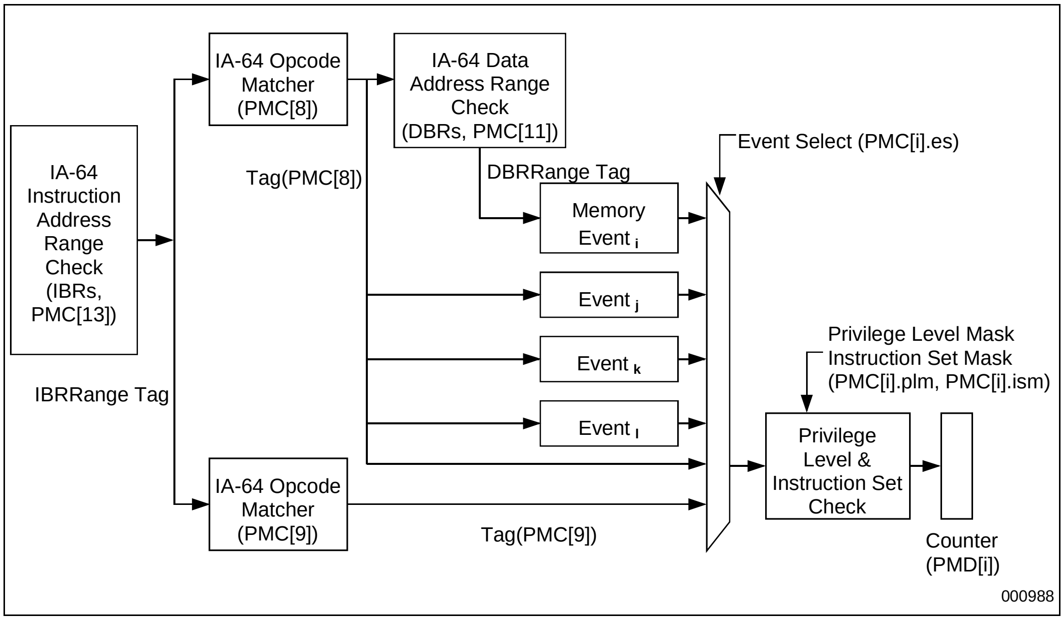
Figure 6-4. Itanium™ Processor Event Qualification



6.1.3.1 Combining Opcode Matching, Instruction, and Data Address Range Check

The Itanium processor allows various event qualification mechanisms to be combined by providing the instruction tagging mechanism shown in Figure 6-5. Instruction address range check and opcode matching are available only for IA-64 code; they are disabled when IA-32 code is executing.

Figure 6-5. Instruction Tagging Mechanism in the Itanium™ Processor



During IA-64 instruction execution (PSR.is is zero), the instruction address range check is applied first. The resulting address range check tag (IBRRangeTag) is passed to two opcode matchers that combine the instruction address range check with the opcode match. Each of the two combined tags (Tag(PMC[8]) and Tag(PMC[9])) can be counted as a retired instruction count event (for details refer to event description IA64_TAGGED_INSTRS_RETIRED in Table 7-3 “Instruction Issue and Retirement Events” on page 7-2).

One of the combined IA-64 address range and opcode match tags, Tag(PMC[8]), qualifies all down-stream pipeline events. Events in the memory hierarchy (L1 and L2 data cache and data TLB events) can further be qualified using a data address DBRRangeTag).

As summarized in Table 6-3, data address range checking can be combined with opcode matching and instruction range checking on the Itanium processor. Additional event qualifications based on the current privilege level and the current instruction set can be applied to all events and are discussed in Section 6.1.3.2, "Privilege Level Constraints" and Section 6.1.3.3, "Instruction Set Constraints".

Table 6-3. Itanium™ Processor Event Qualification Modes

Event Qualification Modes	Instr. Address Range Check PMC[13].ta	Opcode Matching PMC[8]	Data Address Range Check PMC[11].pt
Unconstrained Monitoring (all events)	1	0xffff_ffff_ffff_ffff	1
Instruction Address Range Check only	0	0xffff_ffff_ffff_ffff	1
Opcode Matching only	1	Desired Opcodes	1
Data Address Range Check only	1	0xffff_ffff_ffff_ffff	0
Instruction Address Range Check and Opcode Matching	0	Desired Opcodes	1

Table 6-3. Itanium™ Processor Event Qualification Modes (Continued)

Event Qualification Modes	Instr. Address Range Check PMC[13].ta	Opcode Matching PMC[8]	Data Address Range Check PMC[11].pt
Instruction and Data Address Range Check	0	0xffff_ffff_ffff_ffff	0
Opcode Matching and Data Address Range Check	1	Desired Opcodes	0

6.1.3.2 Privilege Level Constraints

Performance monitoring software cannot always count on context switch support from the operating system. In general, this has made performance analysis of a single process in a multi-processing system or a multi-process workload very difficult. To provide hardware support for this kind of analysis, IA-64 specifies three global bits (PSR.up, PSR.pp, DCR.pp) and a per-monitor “privilege monitor” bit (PMC[i].pm). To break down the performance contributions of operating system and user-level application components, each monitor specifies a 4-bit privilege level mask (PMC[i].plm). The mask is compared to the current privilege level in the processor status register (PSR.cpl), and event counting is enabled if PMC[i].plm[PSR.cpl] is one. The Itanium processor performance monitors control is discussed in Section 6.2.1, "Performance Monitor Control and Accessibility".

PMC registers can be configured as user-level monitors (PMC[i].pm is zero) or system-level monitors (PMC[i].pm is one). A user-level monitor is enabled whenever PSR.up is one. PSR.up can be controlled by an application using the `sum`/`rum` instructions. This allows applications to enable/disable performance monitoring for specific code sections. A system-level monitor is enabled whenever PSR.pp is one. PSR.pp can be controlled at privilege level 0 only, which allows monitor control without interference from user-level processes. The pp field in the default control register (DCR.pp) is copied into PSR.pp whenever an interruption is delivered. This allows events generated during interruptions to be broken down separately: if DCR.pp is zero, events during interruptions are not counted, if DCR.pp is one, they are included in the kernel counts.

As shown in Figure 6-6, Figure 6-7 and Figure 6-8, single process, multi-process, and system level performance monitoring are possible by specifying the appropriate combination of PSR and DCR bits. These bits allow performance monitoring to be controlled entirely from a kernel level device driver, without explicit operating system support. Once the desired monitoring configuration has been setup in a process’ processor status register (PSR), “regular” unmodified operating context switch code automatically enables/disables performance monitoring.

With support from the operating system, individual per-process break-down of event counts can be generated as outlined in Section 7.2, "Performance Monitoring" of Volume 2.

6.1.3.3 Instruction Set Constraints

On the Itanium processor, monitoring can additionally be constrained based on the currently executing instruction set as defined by PSR.is. This capability is supported by the four generic performance counters as well as the instruction and data event address registers. However, the IA-64 instruction address range checking, IA-64 opcode matching and the IA-64 branch trace buffer, only support IA-64 code execution. When these IA-64 only features are used, the corresponding PMC register instruction set mask (PMC[i].ism) should be set to IA-64 only (01) to ensure that events generated by IA-32 code do not corrupt the IA-64 event counts.

Figure 6-6. Single Process Monitor

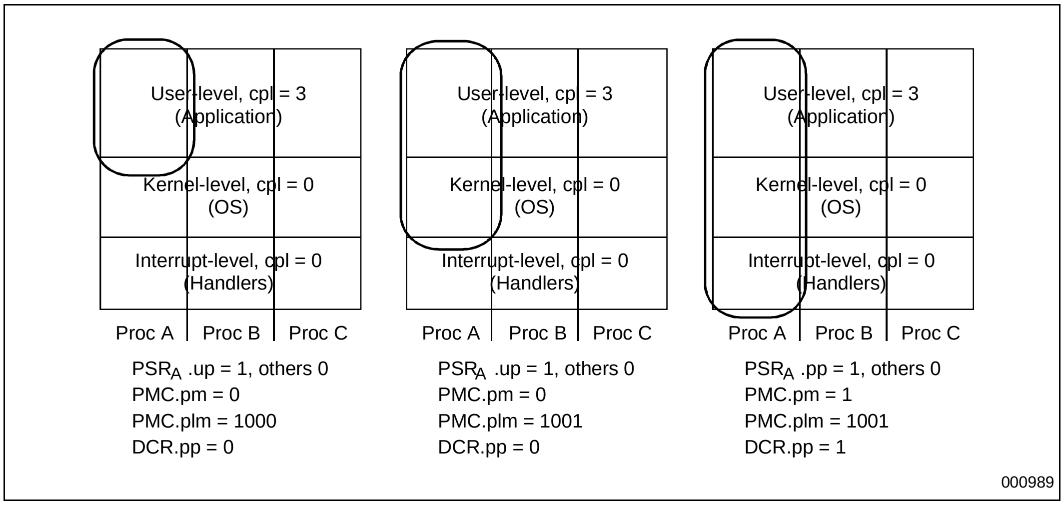


Figure 6-7. Multiple Process Monitor

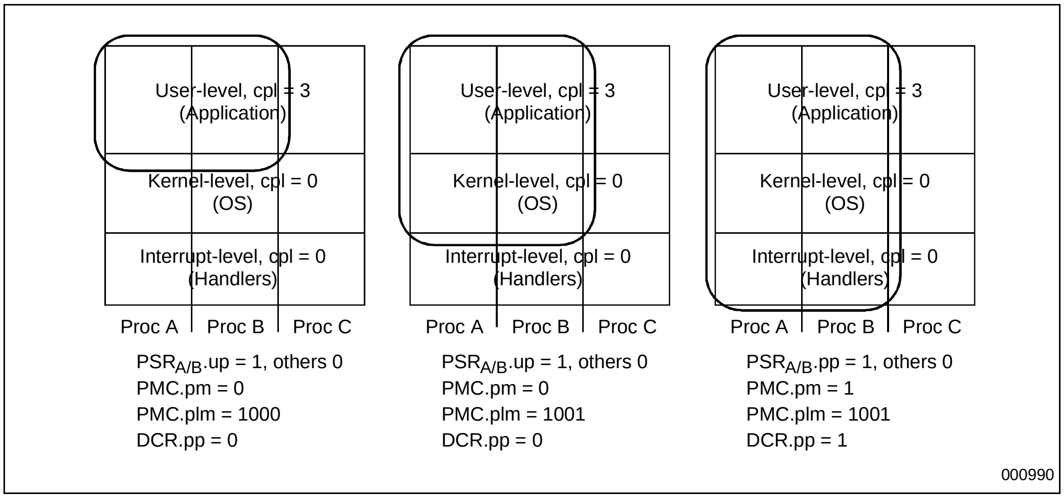
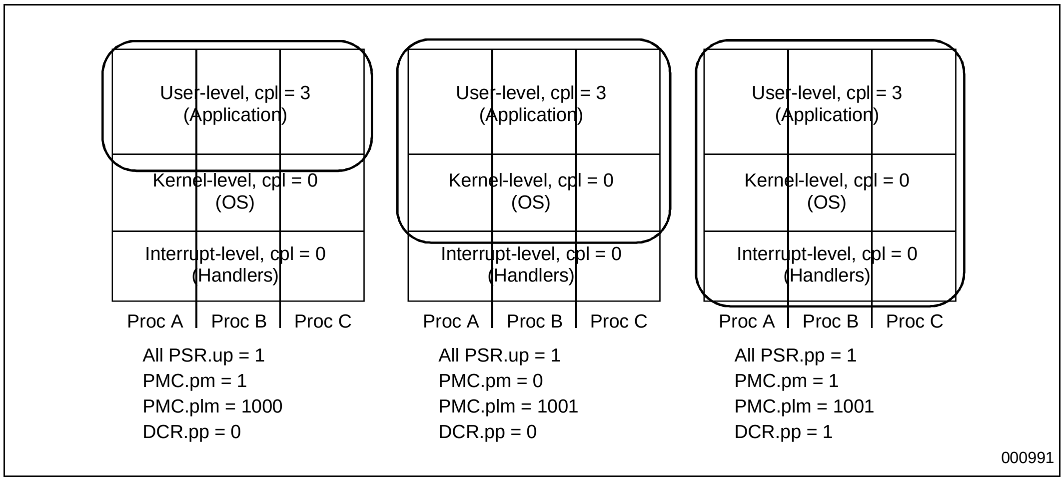


Figure 6-8. System Wide Monitor



6.2 Performance Monitor State

Two sets of performance monitor registers are defined. Performance Monitor Configuration (PMC) registers are used to configure the monitors. Performance Monitor Data (PMD) registers provide data values from the monitors. This section describes the Itanium processor performance monitoring registers which expands on the IA-64 architectural definition. As shown in Figure 6-9, the Itanium processor provides four 32-bit performance counters (PMC/PMD[4,5,6,7] pairs), and the following model-specific monitoring registers: instruction and data event address registers (EARs) for monitoring cache and TLB misses, a branch trace buffer, two opcode match registers and an instruction address range check register.

Table 6-4 defines the PMC/PMD register assignments for each monitoring feature. The interrupt status registers are mapped to PMC[0,1,2,3]. The four generic performance counter pairs are assigned to PMC/PMD[4,5,6,7]. The event address registers and the branch trace buffer are controlled by three configuration registers (PMC[10,11,12]). Captured event addresses and cache miss latencies are accessible to software through five event address data registers (PMD[0,1,2,3,17]) and a branch trace buffer (PMD[8-16]). On the Itanium processor, monitoring of some events can additionally be constrained to a programmable instruction address range by appropriate setting of the instruction breakpoint registers (IBR) and the instruction address range check register (PMC[13]). Two opcode match registers (PMC[8,9]) allow monitoring of some events to be qualified with a programmable opcode. For memory operations, events can be qualified by a programmable data address range by appropriate setting of the data breakpoint registers (DBR) and the data address range check bits in PMC[11].

6.2.1 Performance Monitor Control and Accessibility

Event collection is controlled by the Performance Monitor Configuration (PMC) registers and the processor status register (PSR). Four PSR fields (PSR.up, PSR.pp, PSR.cpl and PSR.sp) and the performance monitor freeze bit (PMC[0].fr) affect the behavior of all performance monitor registers.

Finer, per monitor, control is provided by three PMC register fields (PMC[i].plm, PMC[i].ism, and PMC[i].pm). Instruction set masking based on PMC[i].ism is an Itanium processor model-specific feature. Event collection for a monitor is enabled under the following constraints on the Itanium processor:

```
Monitor Enable_i =(not PMC[0].fr) and PMC[i].plm[PSR.cpl] and ((not
PMC[i].ism[PSR.is]) or (PMC[i]=12)) and (not (PMC[i].pm) and PSR.up) or (PMC[i].pm
and PSR.pp))
```

Figure 3-2, “Processor Status Register (PSR)” on page 3-6 in Volume 2 defines the PSR control fields that affect performance monitoring. For a detailed definition of how the PSR bits affect event monitoring and control accessibility of PMD registers, please refer to Section 3.3.2, "Processor Status Register (PSR)" and Section 7.2.1, "Generic Performance Counter Registers" in Volume 2.

Figure 6-9. Itanium™ Processor Performance Monitor Register Model

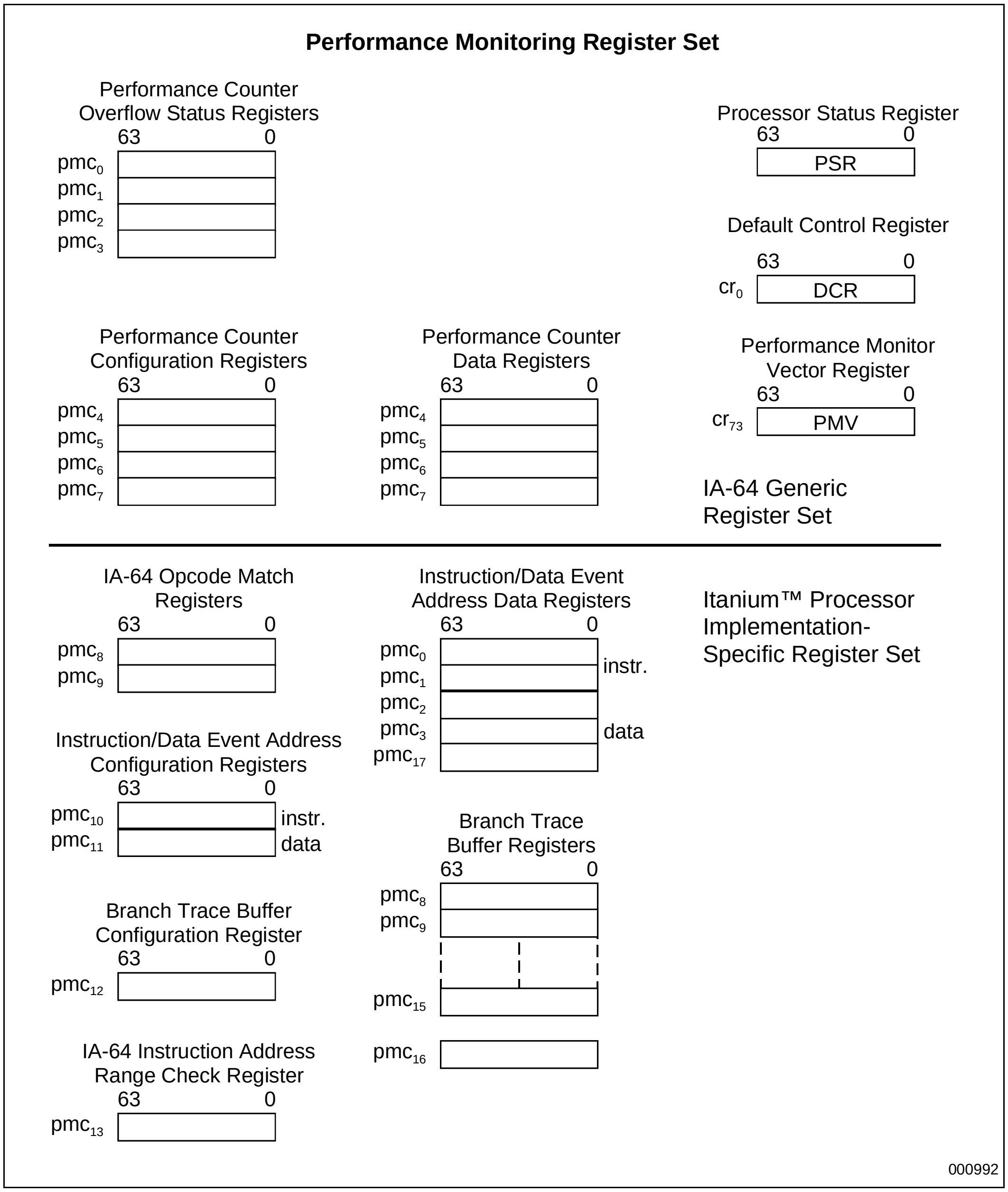


Table 6-4. Itanium™ Processor Performance Monitor Register Set

Monitoring Feature	Configu-ration Registers (PMC)	Data Registers (PMD)	Description
Interrupt Status	PMC[0,1,2,3]	none	See Section 6.2.3, "Performance Monitor Overflow Status Registers (PMC[0,1,2,3])"
Event Counters	PMC[4,5,6,7]	PMD[4,5,6,7]	See Section 6.2.2, "Performance Counter Registers"
Opcode Matching	PMC[8,9]	none	See Section 6.2.5, "IA-64 Opcode Match Registers (PMC[8,9])"
Instruction EAR	PMC[10]	PMD[0,1]	See Section 6.2.7.1, "Instruction EAR (PMC[10], PMD[0,1])"
Data EAR	PMC[11]	PMD[2,3,17]	See Section 6.2.7.4, "Data EAR (PMC[11], PMD[2,3,17])"
Instruction Address Range Check	PMC[13]	none	See Section 6.2.4, "IA-64 Instruction Address Range Check Register (PMC[13])"
Data Address Range Check	PMC[11]	none	See Section 6.2.6, "IA-64 Data Address Range Check (PMC[11])"

As defined in Table 6-4, each of these PMC registers controls the behavior of its associated performance monitor data registers (PMD). Table 6-5 defines per monitor controls that apply to PMC[4,5,6,7,10,11,12]. The Itanium processor model-specific PMD registers associated with instruction/data EARs and the branch trace buffer (PMD[0,1,2,3,8-17]) can be read reliably only when event monitoring is frozen (PMC[0].fr is one).

Figure 6-10. Processor Status Register (PSR) Fields for Performance Monitoring

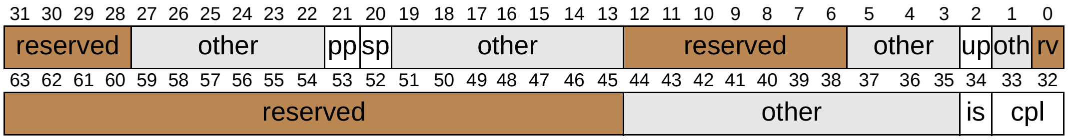


Table 6-5. Performance Monitor PMC Register Control Fields (PMC[4,5,6,7,10,11,12])

Field	Bits	Description
plm	3:0	Privilege Level Mask - controls performance monitor operation for a specific privilege level. Each bit corresponds to one of the 4 privilege levels, with bit 0 corresponding to privilege level 0, bit 1 with privilege level 1, etc. A bit value of 1 indicates that the monitor is enabled at that privilege level. Writing zeros to all plm bits effectively disables the monitor. In this state, the Itanium™ processor will not preserve the value of the corresponding PMD register(s).
pm	6	Privileged monitor - When 0, the performance monitor is configured as a user monitor, and enabled by PSR.up. When PMC.pm is 1, the performance monitor is configured as a privileged monitor, enabled by PSR.pp, and PMD can only be read by privileged software.
ism	25:24	Instruction Set Mask - controls performance monitor operation based on the current instruction set. The instruction set mask applies to PMC[4,5,6,7,10,11] but not to PMC[12]. 00: monitoring enabled during IA-64 and IA-32 instruction execution (regardless of PSR.is) 10: bit 24 low enables monitoring during IA-64 instruction execution (when PSR.is is zero) 01: bit 25 low enables monitoring during IA-32 instruction execution (when PSR.is is one) 11: disables monitoring

6.2.2 Performance Counter Registers

The Itanium processor provides four generic performance counters (PMC/PMD[4,5,6,7] pairs). The implemented counter width on the Itanium processor is 32 bits. The Itanium processor counters are not symmetrical (i.e. not all event types can be monitored by all counters). Counters PMC/PMD[4,5] can track events whose maximum per-cycle event increment is 7. Counters PMC/PMD[6,7] can track events whose maximum per-cycle event increment is 3.

The Itanium processor extends the generic IA-64 counter configuration register (PMC) layout by adding two fields for specifying a unit mask (umask) and a threshold field. These model-specific fields are described in Table 6-6. A counter overflow occurs when the counter wraps (i.e. a carry out from bit 31 is detected). Software can force an external interruption or external notification after N events, by preloading the monitor with a count value of 2^{32} - N. When accessible, software can continuously read the performance counter registers PMD[4,5,6,7] without disabling event collection. The processor guarantees that software will see monotonically increasing counter values.

Figure 6-11 and Table 6-6 define the layout of the Itanium processor Performance Counter Data Registers (PMD[4,5,6,7]). Figure 6-12, Figure 6-13 and Table 6-6 define the layout of the Itanium processor Performance Counter Configuration Registers (PMC[4,5,6,7]).

Figure 6-11. Itanium™ Processor Generic PMD Registers (PMD[4,5,6,7])

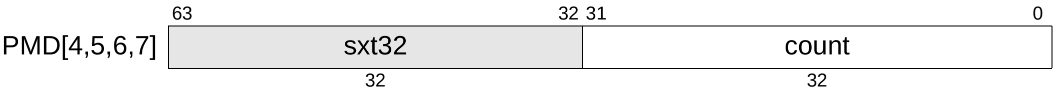


Table 6-6. Itanium™ Processor Generic PMD Register Fields

Field	Bits	Description
sxt32	63:32	Writes are ignored, Reads return the value of bit 31, so count values appear as sign extended.
count	31:0	Event Count. The counter is defined to overflow when the count field wraps (carry out from bit 31).

Figure 6-12. Itanium™ Processor Generic PMC Registers (PMC[4,5])

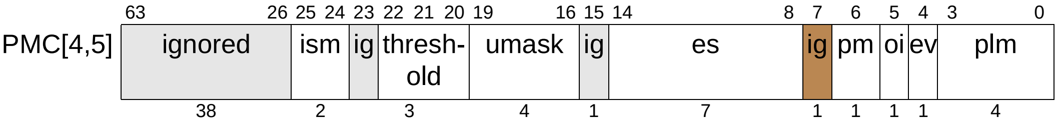


Figure 6-13. Itanium™ Processor Generic PMC Registers (PMC[6,7])

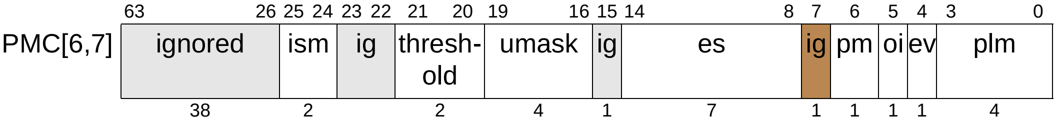


Table 6-7. Itanium™ Processor Generic PMC Register Fields (PMC[4,5,6,7])

Field	Bits	Description
plm	3:0	Privilege Level Mask. See Table 6-5, “Performance Monitor PMC Register Control Fields (PMC[4,5,6,7,10,11,12])”.
ev	4	External visibility - When 1, an external notification (BPM pin strobe) is provided whenever the counter wraps, i.e a carry out from bit 31 is detected. External notification occurs regardless of the setting of the oi bit. On the Itanium™ processor, PMC[4] external notification strobes the BPM0 pin, PMC[5] external notification strobes the BPM1 pin, PMC[6] external notification strobes the BPM2 pin, and PMC[7] external notification strobes the BPM3 pin.
oi	5	Overflow interrupt - When 1, a Performance Monitor Interrupt is raised and the performance monitor freeze bit (PMC[0].fr) is set when the monitor overflows. When 0, no interrupt is raised and the performance monitor freeze bit (PMC[0].fr) remains unchanged. Overflow occurs when the counter wraps, i.e. a carry out from bit 31 is detected. Counter overflows generate only one interrupt.
pm	6	Privilege Monitor. See Table 6-5, “Performance Monitor PMC Register Control Fields (PMC[4,5,6,7,10,11,12])”.
ig	7	ignored
es	14:8	Event select - selects the performance event to be monitored. Itanium processor event encodings are defined in Chapter 7, "Performance Monitor Events".
ig	15	ignored
umask	19:16	Unit Mask - event specific mask bits (see event definition for details)
threshold	22:20 21:20	Threshold -enables thresholding for “multi-occurrence” events. PMC[4,5] define 3 threshold bits 22:20, while PMC[6,7] define 2 threshold bits 21:20. When threshold is zero, the counter sums up all observed event values. When the threshold is non-zero, the counter increments by one in every cycle in which the observed event value exceeds the threshold.
ism	25:24	Instruction Set Mask. See Table 6-5, “Performance Monitor PMC Register Control Fields (PMC[4,5,6,7,10,11,12])”.
ignored	63:24	Read zero, Writes ignored.

6.2.3 Performance Monitor Overflow Status Registers (PMC[0,1,2,3])

The Itanium processor supports four counters. As shown in Figure 6-14 and Table 6-8 only PMC[0]{7:4} bits are populated. All other overflow bits are ignored, i.e. they read as zero and ignore writes.

Figure 6-14. Itanium™ Processor Performance Monitor Overflow Status Registers (PMC[0,1,2,3])

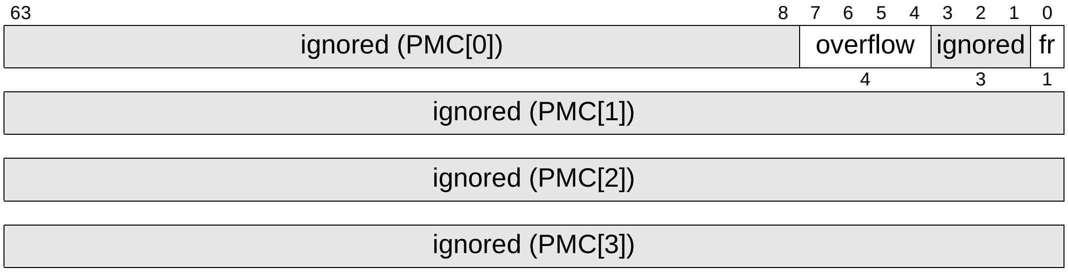


Table 6-8. Itanium™ Processor Performance Monitor Overflow Register Fields (PMC[0,1,2,3])

Register	Field	Bits	Description
PMC[0]	fr	0	Performance Monitor “freeze” bit - when 1, event monitoring is disabled. When 0, event monitoring is enabled. This bit is set by hardware whenever a performance monitor overflow occurs and its corresponding overflow interrupt bit (PMC.oi) is set to one. SW is responsible for clearing it. When the PMC.oi bit is not set, then counter overflows do not set this bit.
PMC[0]	ignored	3:1	Read zero, Writes ignored.
PMC[0]	overflow	7:4	Event Counter Overflow - When bit n is one, indicate that the PMD_n overflowed. This is a bit vector indicating which performance monitor overflowed. These overflow bits are set on their corresponding counters overflow regardless of the state of the PMC.oi bit. These bits are sticky and multiple bits may be set.
PMC[0]	ignored	63:8	Read zero, Writes ignored.
PMC [1,2,3]	ignored	63:0	Read zero, Writes ignored.

6.2.4 IA-64 Instruction Address Range Check Register (PMC[13])

The Itanium processor allows event monitoring to be constrained to a range of instruction addresses. All four architectural breakpoint registers (IBRs) are used to specify the desired address range. The Itanium processor instruction address range check register PMC[13] specifies how the resulting address match is applied to the performance monitors.

Figure 6-15. Itanium™ Processor Instruction Address Range Check Register (PMC[13])

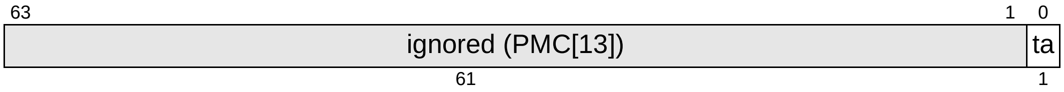


Table 6-9. Itanium™ Processor Instruction Address Range Check Register Fields (PMC[13])

Field	Bits	Description
ta	0	Tag All - when 1, all events are counted independent of instruction address and instruction set. The default value of this PMC[13].ta should be set to one upon reset.

Instruction address range checking is controlled by the “tag all” bit (PMC[13].ta). When PMC[13].ta is one, all instructions are tagged regardless of IBR settings. In this mode, events from both IA-32 and IA-64 code execution contribute to the event count. When PMC[13].ta is zero, the instruction address range check based on the IBR settings is applied to all IA-64 code fetches. In this mode, IA-32 instructions are never tagged, and, as a result, events generated by IA-32 code execution are ignored. Table 6-10 defines the behavior of the instruction address range checker for different combinations of PSR.is and PMC[13].ta.

Table 6-10. Itanium™ Processor Instruction Address Range Check by Instruction Set

	PSR.is	
PMC_{13}.ta	0 (IA-64)	1 (IA-32)
0	Tag only IA-64 instructions if they match IBR range	DO NOT tag any IA-32 operations.
1	Tag all IA-64 and IA-32 instructions.Ignore IBR range.	

The processor compares every IA-64 instruction fetch address IP{63:0} with each of the four architectural instruction breakpoint registers. Regardless of the value of the instruction break-point fault enable (IBR x-bit), the following expression is evaluated for each of the Itanium processor’s four IBRs:

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

On the Itanium processor, in which only 54 virtual and 44 physical address bits are implemented, this IBR match is defined as follows:

$IBRmatch_i = (IBR_{[2*i]+1}.plm[PSR.cpl])$
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 $(AND_{b=63:61}$ $(IBR_i.addr\{b\}=IP\{b\}))$

The resulting four matches are combined with the PSR.is bit, two instruction address range check register bits, the IBR x-bits, and PSR.db:

IBRRangeTag = (PMC[13].ta)
or ((not PSR.is)
and $((IBRmatch_0$ or $IBRmatch_1$ or $IBRmatch_2$ or $IBRmatch_3)$
and (not (PSR.db or IBR_1.x or IBR_3.x or IBR_5.x or IBR_7.x))))

The instruction range check tag (IBRRangeTag) considers the IBR address ranges only if PMC[13].ta is zero, PSR.is is zero, and if none of the IBR x-bits or PSR.db are set. Since the architectural break-point registers (IBRs) are used to specify the desired performance monitor address range, it is not possible to constrain monitoring when the IBRs are used in their architectural break-point capacity, i.e. when PSR.db or an IBR x-bit is set. In other words, it is not possible to use performance monitor address range checking when a debugger is running, unless the debugger and the performance monitor software carefully synchronize their use of the IBRs.

The instruction range check tag is computed early in the processor pipeline and therefore includes speculative, wrong-path as well as predicated off instructions. Furthermore, range check tags are not accurate in the instruction fetch and out-of-order parts of the pipeline (cache and bus units). Therefore, software must accept a level of range check inaccuracy for events generated by these units, especially for non-looping code sequences that are shorter than the Itanium processor pipeline. As described in Section 6.1.3.1, "Combining Opcode Matching, Instruction, and Data Address Range Check", the instruction range check result may be combined with the results of the IA-64 opcode match registers described in the next section.

6.2.5 IA-64 Opcode Match Registers (PMC[8,9])

The Itanium processor allows event monitoring to be constrained based on the IA-64 encoding (opcode) of an instruction. Registers PMC[8,9] allow two independent opcodes matches to be specified. The IA-64 opcode matcher operates only during IA-64 code execution (i.e. when PSR.is is zero).

Figure 6-16. Opcode Match Registers (PMC[8,9])

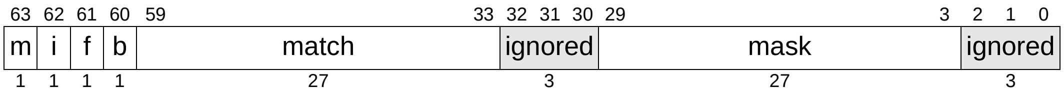


Table 6-11. Opcode Match Register Fields (PMC[8,9])

Field	Bits	Width	Description
mask	29:3	27	Bits that mask IA-64 instruction encoding bits {40:27} and {12:0}
match	59:33	27	Opcode bits to match IA-64 instruction encoding bits {40:27} and {12:0}
b	60	1	If 1: match if opcode is an B-syllable
f	61	1	If 1: match if opcode is an F-syllable
i	62	1	If 1: match if opcode is an I-syllable
m	63	1	If 1: match if opcode is an M-syllable

For opcode matching purposes, an IA-64 instruction is defined by two items: the instruction type “itype” (one of M, I, F or B) and the 40-bit encoding “enco{40:0}” defined in Volume 3. Each instruction is evaluated against each opcode match register (PMC[8,9]) as follows:

Match(PMC[i]) = (imatch (itype,PMC[i].mifb) and
ematch(enco,PMC[i].match,PMC[i]PMC[i].mask))

Where:

imatch(itype,PMC[i].mifb) = itype=M and PMC[i].m) or (itype=I and PMC[i].i) or (itype=F and PMC[i].f) or (itype=B and PMC[i].b)

ematch(enco,match,mask) = AND $_{b=40..27}$ ((enco{b}=match{b-14}) or mask{b-14}) and AND $_{b=12..0}$ ((enco{b}=match{b}) or mask{b})

This function matches encoding bits{40:27} (major opcode) and encoding bits{12:0} (destination and qualifying predicate) only. Bits{26:13} of the instruction encoding are ignored by the opcode matcher.

This produces two opcode match events that are combined with the PSR.is bit, and the instruction range check tag (IBRRangeTag, see Section 6.2.4, "IA-64 Instruction Address Range Check Register (PMC[13])") as follows:

Tag(PMC[8]) = Match(PMC[8]) and IBRRangeTag and (not PSR.is)

Tag(PMC[9]) = Match(PMC[9]) and IBRRangeTag and (not PSR.is)

As shown in Figure 6-5, the two tags, Tag(PMC[8]) and Tag(PMC[9]), are staged down the processor pipeline until instruction retirement, and can be selected as a retired instruction count event. In this way, a performance counters (PMC/PMD[4,5,6,7]) can be used to count the number of retired instructions within the programmed range that match the specified opcodes. All combinations of the mifb bits are supported. To match A-syllable instructions both m and i bits should be set to one. To match all instruction types, all mifb and all mask bits should be set to one. This will count the number of retired instructions within the programmed address range. One of the combined IA-64 address range and opcode match tags, Tag(PMC[8]), qualifies most down-stream pipeline events. To ensure that all events are counted independent of the IA-64 opcode matcher, all mifb and all mask bits of PMC[8] should be set to one (all opcodes match). Tag(PMC[9]) is not used to qualify downstream events.

6.2.6 IA-64 Data Address Range Check (PMC[11])

For instructions that reference memory, the Itanium processor allows event counting to be constrained by data address ranges using the architectural data breakpoint registers (DBRs). Data address range checking capability is controlled enabled by the “pass tags” bit in the Data Event Address Register (PMC[11].pt). For details on PMC[11], refer to Section 6.2.7.4, "Data EAR (PMC[11], PMD[2,3,17])".

When enabled (PMC[11].pt is zero), data address range checking is applied to loads (all types), stores, semaphore operations, and the `lfetch` instruction whose upstream opcode match Tag(PMC[8]) was set. When PMC[11].pt is one, RSE operations and VHPT walks are tagged only if the opcode match Tag(PMC[8]) was set for the operation that caused the RSE or VHPT activity. When PMC[11].pt is zero, all RSE operations and VHPT walks that hit the programmed data address range are tagged (regardless of the opcode match Tag(PMC[8])). To capture all VHPT walks when PMC[11].pt is zero, the minimum DBR mask granularity must be set to the size of a single VHPT entry.

On the Itanium processor, in which only 54 virtual address bits are implemented, the performance monitoring DBR match function is defined as follows:

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}))

The resulting four matches are combined with PSR.db to form a single DBR match:

$\text{DBRRangeMatch} = ((\text{DBRRangeMatch}_0 \text{ or DBRRangeMatch}_1 \text{ or DBRRangeMatch}_2 \text{ or}$
$\text{DBRRangeMatch}_3)$
and (not PSR.db))

Note: DBR matching for performance monitoring ignores the setting of the DBR r, w and plm fields. Finally, the DBRRangeMatch is combined with PMC[11].pt and the upstream opcode match tag Tag(PMC[8]) as follows:

DBRRangeTag = Tag(PMC[8]) and ((PMC[11].pt) or DBRRangeMatch)

DBR based data address range checking combined with opcode matching and instruction range checking allows the following combinations of event monitoring on the Itanium processor.

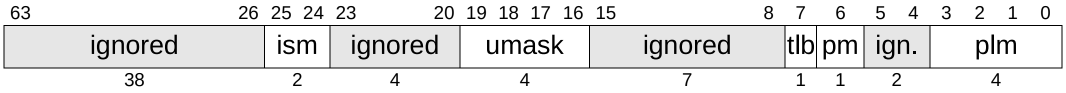
6.2.7 Event Address Registers (PMC[10,11]/PMD[0,1,2,3,17])

This section defines the register layout for the Itanium processor instruction and data event address registers (EARs). Sampling of four events is supported on the Itanium processor: instruction cache and instruction TLB misses, data cache load misses, and data TLB misses. The EARs are configured through two PMC registers (PMC[10,11]). EAR specific unit masks allow software to specify event collection parameters to hardware. Instruction and data addresses, operation latencies and other captured event parameters are provided in five PMD registers (PMD[0,1,2,3,17]). The instruction and data cache EARs report the latency of captured cache events and allow latency thresholding to qualify event capture. Event address data registers (PMD[0,1,2,3,17]) contain valid data only when event collection is frozen (PMC[0].fr is one). Reads of PMD[0,1,2,3,17] while event collection is enabled return undefined values.

6.2.7.1 Instruction EAR (PMC[10], PMD[0,1])

The instruction event address configuration register (PMC[10]) can be programmed to monitor either L1 instruction cache or instruction TLB miss events. Figure 6-17 and Table 6-12 detail the register layout of PMC[10]. Figure 6-18 describes the associated event address data registers PMD[0,1].

Figure 6-17. Instruction Event Address Configuration Register (PMC[10])

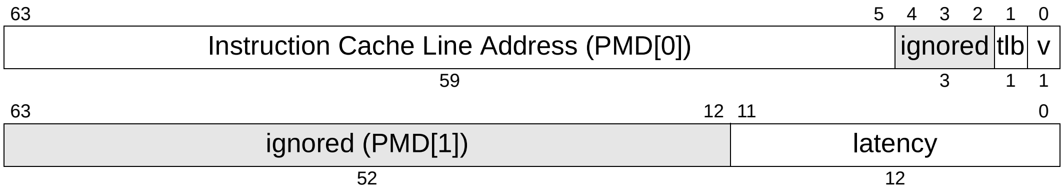


When the tlb-bit (PMC[10].tlb) is set to zero instruction cache misses are monitored, when it is set to one instruction TLB misses are monitored. The interpretation of the umask field and performance monitor data registers PMD[0,1] depend on the setting of the tlb bit, and are described in Section 6.2.7.2, "Instruction EAR Cache Mode (PMC[10].tlb=0)" for instruction cache monitoring and in Section 6.2.7.3, "Instruction EAR TLB Mode (PMC[10].tlb=1)" for instruction TLB monitoring.

Table 6-12. Instruction Event Address Configuration Register Fields (PMC[10])

Field	Bits	Description	
plm	3:0	See Table 6-5.	
pm	6	See Table 6-5.	
tlb	7	Instruction EAR selector: instruction cache/TLB	
		if tlb=0:	monitor L1 instruction cache misses PMD[0,1] register interpretation see Table 6-14.
		if tlb=1:	monitor instruction TLB misses PMD[0,1] register interpretation see Table 6-16.
umask	19:16	Instruction EAR unit mask	
		if tlb=0:	instruction cache unit mask (definition see Table 6-13)
		if tlb=1:	instruction TLB unit mask (definition see Table 6-15)
ism	25:24	See Table 6-5.	

Figure 6-18. Instruction Event Address Register Format (PMD[0,1]



6.2.7.2 Instruction EAR Cache Mode (PMC[10].tlb=0)

When PMC[10].tlb is zero, the instruction event address register captures instruction addresses and access latencies for L1 instruction cache misses. Only misses whose latency exceeds a programmable threshold are captured. The threshold is specified as a four bit umask field in the configuration register PMC[10]. Possible threshold values are defined in Table 6-13.

As defined in Table 6-14, the address of the instruction cache line missed the L1 instruction cache is provided in PMD[0]. If no qualified event was captured, the valid bit in PMD[0] is zero. The latency of the captured instruction cache miss in processor clock cycles is provided in the latency field of PMD[1]. In cache mode, the TLB miss bit of PMD[0] is undefined.

Table 6-13. Instruction EAR (PMC[10]) umask Field in Cache Mode (PMC[10].tlb=0)

umask Bits 3:0	Latency Threshold [CPU cycles]	umask Bits 3:0	Latency Threshold [CPU cycles]
0000	>= 4	0110	>= 256
0001	>= 8	0111	>= 512
0010	>= 16	1000	>= 1024
0011	>= 32	1001	>= 2048
0100	>= 64	1010	>= 4096
0101	>= 128	1011.. 1111	No events are captured.

Table 6-14. Instruction EAR (PMD[0,1]) in Cache Mode (PMC[10].tlb=0)

Register	Field	Bits	Description
PMD[0]	v	0	Valid Bit 0: invalid address (EAR did not capture qualified event) 1: EAR contains valid event data
	tlb	1	TLB Miss Bit (undefined in cache mode)
	Instruction Cache Line Address	63:5	Address of instruction cache line that caused cache miss[a]
PMD[1]	latency	11:0	Latency in processor clocks

a. The Itanium™ processor does not implement virtual address bits va{60:51} and physical address bits pa{62:44}. The instruction and data address bits {60:51} of PMD[0] read as a sign-extension of bit {50}. Writes to bits {60:51} of PMD[0] are ignored by the processor.

6.2.7.3 Instruction EAR TLB Mode (PMC[10].tlb=1)

When PMC[10].tlb is one, the instruction event address register captures addresses of instruction TLB misses. The unit mask allows event address collection to capture specific subsets of instruction TLB misses. Table 6-15 summarizes the instruction TLB umask settings. All combinations of the mask bits are supported.

As defined in Table 6-16, the address of the instruction cache line fetch that missed the L1 TLB is provided in PMD[0]. The tlb bit indicates whether the captured TLB miss hit in the VHPT or required servicing by software. If no qualified event was captured, the valid bit in PMD[0] reads zero. In TLB mode, the latency field of PMD[1] is undefined.

Table 6-15. Instruction EAR (PMC[10]) umask Field in TLB Mode (PMC[10].tlb=1)

umask Bit	Instruction TLB EAR Unit Mask (Instruction TLB misses)
0	ignored
1	ignored
2	if one, capture Instruction TLB misses that hit VHPT
3	if one, capture Instruction TLB misses handled by software

Table 6-16. Instruction EAR (PMD[0,1]) in TLB Mode (PMC[10].tlb=1)

Register	Field	Bits	Description
PMD[0]	v	0	Valid Bit 0: invalid address (EAR did not capture qualified event 1: EAR contains valid event data
	tlb	1	TLB Miss Bit: 0: VHPT Hit 1: Instruction TLB Miss handled by software
	Instruction Cache Line Address	63:5	Address of instruction cache line that caused TLB miss[a]
PMD[1]	latency	11:2	undefined in TLB mode

a. The Itanium™ processor does not implement virtual address bits va{60:51}. The instruction address bits {60:51} of PMD[0] read as a sign-extension of bit {50}. Writes to bits {60:51} of PMD[0] are ignored by the processor.

6.2.7.4 Data EAR (PMC[11], PMD[2,3,17])

The data event address configuration register (PMC[11]) can be programmed to monitor either L1 data cache load misses or L1 data TLB misses. Figure 6-19 and Table 6-17 detail the register layout of PMC[11]. Figure 6-20 describes the associated event address data registers PMD[2,3,17]. The tlb bit in configuration register PMC[11] selects data cache or data TLB monitoring. The interpretation of the umask field and registers PMD[2,3,17] depends on the setting of the tlb bit, and is described in Section 6.2.7.5, "Data Cache Load Miss Monitoring (PMC[11].tlb=0)" for data cache load miss monitoring and in Section 6.2.7.6, "Data TLB Miss Monitoring (PMC[11].tlb=1)" for data TLB monitoring. The PMC[11].pt bit controls data address range checking which is described in Section 6.2.6, "IA-64 Data Address Range Check (PMC[11])".

Figure 6-19. Data Event Address Configuration Register (PMC[11])

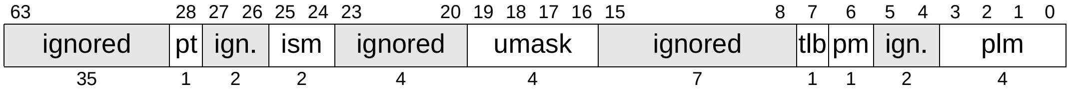
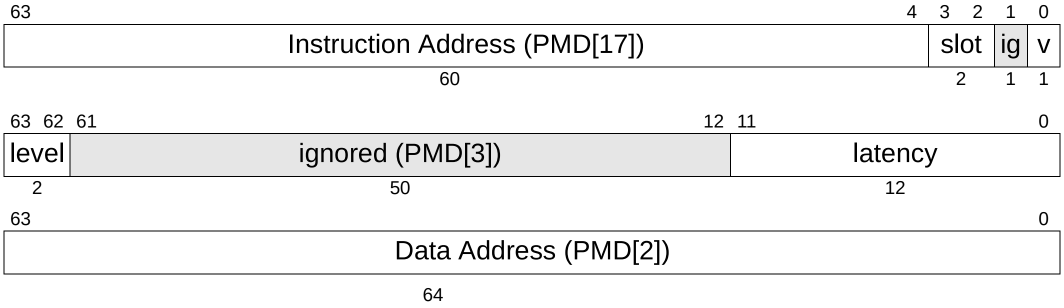


Table 6-17. Data Event Address Configuration Register Fields (PMC[11])

Field	Bits	Description
plm	3:0	See Table 6-5.
pm	6	See Table 6-5.
tlb	7	Data EAR selector: data cache/TLB
		if tlb=0:monitor L1 data cache load misses PMD[2,3,17] register interpretation see Table 6-19.
		if tlb=1: monitor L1 data TLB misses PMD[2,3,17] register interpretation see Table 6-21.
umask	19:16	Data EAR unit mask if tlb=0: data cache unit mask (definition see Table 6-18) if tlb=1: data TLB unit mask (definition see Table 6-20)
ism	25:24	See Table 6-5.
pt	28	Pass Tags. This bit enables/disables data address range checking. See Section 6.2.6, "IA-64 Data Address Range Check (PMC[11])" for details. if pt=1: then the Tag(PMC[8]) is passed down the pipeline unmodified. if pt=0: data address range checking is enabled for memory operations.

Figure 6-20. Data Event Address Register Format (PMD[2,3,17])



6.2.7.5 Data Cache Load Miss Monitoring (PMC[11].tlb=0)

If the Data EAR is configured to monitor data cache load misses (PMC[11].tlb=0), the umask is used as a load latency threshold defined by Table 6-18.

As defined in Table 6-19, the instruction and data addresses as well as the load latency of a captured data cache load miss is presented to software in three registers PMD[2,3,17]. If no qualified event was captured, the valid bit in PMD[3] is zero. In data cache load miss mode, the level field of PMD[3] is undefined.

Table 6-18. PMC[11] Mask Fields in Data Cache Load Miss Mode (PMC[11].tlb=0)

umask Bits 3:0	Latency Threshold [CPU cycles]	umask Bits 3:0	Latency Threshold [CPU cycles]
0000	>= 4	0110	>= 256
0001	>= 8	0111	>= 512
0010	>= 16	1000	>= 1024
0011	>= 32	1001	>= 2048
0100	>= 64	1010	>= 4096
0101	>= 128	1011.. 1111	No events are captured.

Table 6-19. PMD[2,3,17] Fields in Data Cache Load Miss Mode (PMC[11].tlb=0)

Register	Fields	Bit Range	Description
PMD[2]	Data Address	63:0	64-bit address of data item that caused miss[a]
PMD[3]	latency	11:0	Latency in CPU clocks
	level	63:62	Undefined in data cache load miss mode
PMD[17]	valid	0	Valid bit 0: invalid address (EAR did not capture qualified event) 1: EAR contains valid event data
	slot	3:2	Instruction bundle slot of memory instruction. For IA-32 ISA mode, this field is undefined.
	Instruction Address	63:4	Address of bundle that contains memory instruction.[a]

[a] The Itanium™ processor does not implement virtual address bits va{60:51} and physical address bits pa{62:44}. The data/instruction address bits {60:51} of PMD[2,17] read as a sign-extension of bit {50}. Writes to bits {60:51} of PMD[2,17] are ignored by the processor.

The detection of data cache load misses requires a load instruction to be tracked during multiple clock cycles from instruction issue to cache miss occurrence. Since multiple loads may be outstanding at any point in time and the Itanium processor data cache miss event address register can only track a single load at a time, not all data cache load misses may be captured. When the processor hardware captures the address of a load (called the monitored load), it ignores all other overlapped concurrent loads until it is determined whether the monitored load turns out to be an L1 data cache miss or not. If the monitored load turns out to be a cache miss, its parameters are latched into PMD[2,3,17]. The processor randomizes the choice of which load instructions are tracked to prevent the same data cache load miss from always being captured (in a regular sequence of overlapped data cache load misses). While this mechanism will not always capture all data cache load misses in a particular sequence of overlapped loads, its accuracy is sufficient to be used by statistical sampling or code instrumentation.

6.2.7.6 Data TLB Miss Monitoring (PMC[11].tlb=1)

If the Data EAR is configured to monitor data TLB misses (PMC[11].tlb=1), the umask defined by Table 6-20 determine which data TLB misses are captured by the Data EAR. For TLB monitoring, all combinations of the mask bits are supported.

Table 6-20. PMC[11] Unmask Field in TLB Miss Mode (PMC[11].tlb=1)

umask Bit	Data EAR Unit Mask (L1 data TLB misses)
0	reserved
1	if one, capture L1 TLB misses that hit L2 Data TLB
2	if one, capture L1 TLB misses that hit VHPT
3	if one, capture L1 TLB misses that was handled by software

As defined in Table 6-21, the instruction and data addresses of captured data TLB misses are presented to software in PMD[2,17]. The level of the TLB hierarchy from which the L1 data TLB miss was satisfied is recorded in the level field of PMD[3]. If no qualified event was captured, the valid bit in PMD[17] and the level field in PMD[3] read zero. When programmed for data TLB monitoring, the contents of the latency field of PMD[3] are undefined.

Table 6-21. PMD[2,3,17] Fields in TLB Miss Mode (PMC[11].tlb=1)

Register	Field	Bit Range	Description
PMD[2]	Data Address	63:0	64-bit address of data item that caused miss[a]
PMD[3]	latency	11:0	Undefined in TLB Miss mode
	level	63:62	Data TLB Miss Level 0: invalid address (EAR did not capture qualified event) 1: L2 Data TLB hit 2: VHPT hit 3: Data TLB miss handled by software
PMD[17]	valid	0	Valid Bit: 0: invalid address (EAR did not capture qualified event) 1: EAR contains valid event data
	slot	3:2	Instruction Bundle Slot of memory instruction. In IA-32 ISA mode, this field is undefined.
	Instruction Address	63:4	Address of bundle that contains memory instruction[a]

[a] The Itanium™ processor does not implement virtual address bits va{60:51} and physical address bits pa{62:44}. The data/instruction address bits {60:51} of PMD[2,17] read as a sign-extension of bit {50}. Writes to bits {60:51} of PMD[2,17] are ignored by the processor.

6.2.8 IA-64 Branch Trace Buffer

The branch trace buffer provides information about the outcome of the most recent IA-64 branch instructions and their predictions and outcomes. The IA-64 branch trace buffer configuration register (PMC[12]) defines the conditions under which branch instructions are captured and allows the trace buffer to capture specific subsets of branch events. The IA-64 branch trace buffer operates only during IA-64 code execution (i.e. when PSR.is is zero).

In every cycle in which a qualified IA-64 branch retires, its source bundle address and slot number are written to the branch trace buffer. The branches' target address is written to the next buffer location. If the target instruction bundle itself contains a qualified IA-64 branch, the branch trace buffer either records a single trace buffer entry (with the b-bit set) or makes two trace buffer entries:

one that records the target instruction as a branch target (b-bit cleared), and another that records the target instruction as a branch source (b-bit set). As a result, the branch trace buffer may contain a mixed sequence of the branches and targets.

6.2.8.1 IA-64 Trace Buffer Collection Constraining

The IA-64 branch trace buffer configuration register (PMC[12]) defines the conditions under which branch instructions are captured. These conditions are given in Figure 6-21 and Table 6-22, and refer to conditions associated with the branch prediction and resolution hardware. These conditions are:

- Which branch prediction hardware structure made the prediction,
- The path of the branch (not taken/taken),
- Whether or not the branch path was mispredicted, and
- Whether or not the target of the branch was mispredicted.

Figure 6-21. IA-64 Branch Trace Buffer Configuration Register (PMC[12])

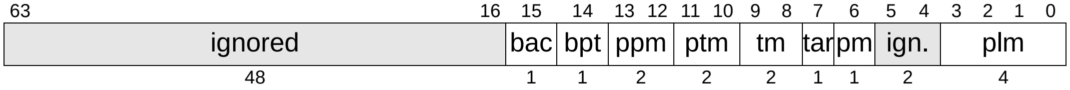


Table 6-22. IA-64 Branch Trace Buffer Configuration Register Fields (PMC[12])

Field	Bits	Description
plm	3:0	See Table 6-5.
pm	6	See Table 6-5.
tar	7	Target Address Register: 1: capture TAR predictions 0: No TAR predictions are captured
tm	9:8	Taken Mask: 11: all IA-64 branches 10: Taken IA-64 branches only 01: Not Taken IA-64 branches only 00: No branch is captured
ptm	11:10	Predicted Target Address Mask: 11: capture branch regardless of target prediction outcome 10: branch predicted target address correctly 01: branch mispredicted target address 00: No branch is captured
ppm	13:12	Predicted Predicate Mask: 11: capture branch regardless of predicate prediction outcome 10: branch predicted branch path (taken/not taken) correctly 01: branch mispredicted branch path (taken/not taken) 00: No branch is captured
bpt	14	Branch Prediction Table: 10: No TAC predictions are captured
bac	15	Branch Address Calculator: 1: capture BAC predictions 0: No BAC predictions are captured

The Itanium processor uses the following micro-architectural structures for branch prediction: the Target Address Registers (TAR), and Target Address Cache (TAC). Using the tar and bac fields of the branch trace buffer configuration register (PMC[12]), collection in the branch trace buffer can be restricted to only branches predicted by a subset of these prediction structures.

The Target Address Registers (TAR) are a small and fast fully associative buffer that is exclusively written to by branch predict instructions with the '.imp' extension. A hit in the TAR will cause a taken prediction and yield the target address of the branch. If the tar field in the branch trace buffer configuration register (PMC[12]) is set to one, branches predicted by TAR will be included in the trace buffer.

The Target Address Cache (TAC) is a larger structure that is also written to by branch predict instructions, or the prediction hardware. The primary function of the TAC is to provide the target address of a branch.

If the bpt field in the branch trace buffer configuration register (PMC[12]) is set to one, branches predicted by the TAC will be included in the trace buffer.

If neither the TAR nor TAC generated a hit, the branch has to be predicted using the static hints encoded in the branches and the target address has to be calculated. This is done by the branch address corrector (BAC). If the bac field in the branch trace buffer configuration register (PMC[12]) is set to one, branches predicted by the branch address corrector will be included in the trace buffer.

Furthermore, using the ptm, ppm and tm fields in the branch trace buffer configuration register (PMC[12]) collection in the branch trace buffer can be restricted based on the correctness of target and predicate prediction in addition to whether the branch was actually taken or not.

To summarize, an IA-64 branch and its target are captured by the trace buffer if the following equation is true:

```
(not PSR.is)
and (       (tm[1] and branch taken)
       or (tm[0] and branch not taken)
    )
and (       (ptm[1] and hardware predicted target address correctly
                    and hardware predicted the branch path correctly
                    and branch is taken)
       or  (ptm[0] and hardware mispredicted target address
                    and hardware predicted the branch path correctly
                    and branch is taken)
       or (ptm[0] and ptm[1])
    )
and (       (ppm[1] and hardware predicted the branch path correctly)
       or  (ppm[0] and hardware mispredicted the branch path)
    )
and (       (bpt and branch was predicted by TAC)
       or  (bac and branch was predicted by BAC)
       or  (tar and branch was predicted by TAR)
    )
```

To capture all mispredicted IA-64 branches, the branch trace buffer configuration settings in PMC[12] should be: Tm=11, ptm=01, ppm=01, bpt=1, bac=1, and tar=1.

6.2.8.2 IA-64 Branch Trace Buffer Reading

The eight branch trace buffer registers PMD[8-15] provide information about the outcome of a captured branch sequence. The branch trace buffer registers (PMD[8-15]) contain valid data only when event collection is frozen (PMC[0].fr is one). While event collection is enabled, reads of PMD[8-15] return undefined values. The registers follow the layout defined in Figure 6-22, and contain the address of either a captured branch instruction (b-bit=1) or branch target (b-bit=0). For branch instructions, the mp-bit indicates a branch misprediction. A branch trace register with a zero b-bit and a zero mp-bit indicates an invalid branch trace buffer entry. The slot field captures the slot number of the first taken IA-64 branch instruction in the captured instruction bundle. A slot number of 3 indicates a not-taken branch. The target address bundle of a branch to IA-32 (`br.ia`) is recorded. An IA-32 JMPE branch instruction and its IA-64 target are not recorded.

Figure 6-22. Branch Trace Buffer Register Format (PMD[8-15])

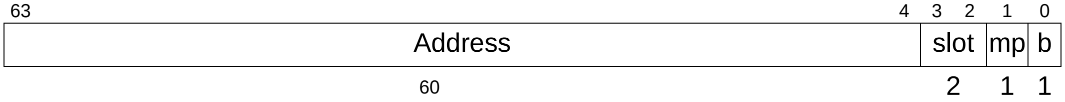


Table 6-23. IA-64 Branch Trace Buffer Register Fields (PMD[8-15])

Field	Bit Range	Description
b	0	Branch Bit 1: contents of register is a branch instruction 0: contents of register is a branch target
mp	1	Mispredict Bit if b=1 and mp=1: mispredicted branch (due to target or predicate misprediction) if b=1 and mp=0: correctly predicted branch if b=0 and mp=0: **invalid branch trace buffer register** if b=0 and mp=1: valid target address
slot	3:2	if b=0: 00 if b=1: Slot index of first taken branch instruction in bundle 00: IA-64 Slot 0 branch/target 01: IA-64 Slot 1 branch/target 10: IA-64 Slot 2 branch/target 11: this was a not taken branch
Address	63:4	if b=1: 60-bit bundle address of IA-64 branch instruction[a] if b=0: 60-bit target bundle address of IA-64 branch instruction[a]

[a] The Itanium™ processor does not implement virtual address bits va{60:51} and physical address bits pa{62:44}. When the processor captures an instruction address, bits {60:51} of PMD[8-15] are written by the processor with a sign-extension of bit {50} of the captured address. When PMD[8-15] are written by software bits {60:51} of PMD[8-15] can be written with any value (not necessarily a sign-extension of bit {50}).

In every cycle in which a qualified IA-64 branch retires[1], its source bundle address and slot number are written to the branch trace buffer. The branches' target address is written to the next buffer location. If the target instruction bundle itself contains a qualified IA-64 branch, the branch trace buffer either records a single trace buffer entry (with the b-bit set) or makes two trace buffer entries:

---

[1] In some cases, the Itanium™ processor branch trace buffer will capture the source (but not the target) address of an excepting branch instruction. This occurs on trapping branch instructions as well as faulting `br.ia`, `break.b` and multiway branches.

one that records the target instruction as a branch target (b-bit cleared), and another that records the target instruction as a branch source (b-bit set). As a result, the branch trace buffer may contain a mixed sequence of the branches and targets.

The IA-64 branch trace buffer is a circular buffer containing the last four to eight qualified IA-64 branches. The Branch Trace Buffer Index Register (PMD[16]) defined in Figure 6-23 identifies the most recently recorded branch or target. In every cycle in which a qualified branch (branch or target) is recorded, the branch buffer index (bbi) is post-incremented. After 8 entries have been recorded, the branch index wraps around, and the next qualified branch will overwrite the first trace buffer entry. The wrap condition itself is recorded in the full bit of PMD[16]. The bbi field of PMD[16] defines the next branch buffer index that is about to be written.The following formula computes the last written branch trace buffer PMD index from the contents of PMD[16]:

last-written-PMD-index = 8+ ([(8*PMD[16].full) + (PMC[16].bbi - 1)] % 8)

If both the full bit and the bbi field of PMD[16] are zero, no qualified branch has been captured by the branch trace buffer. The full bit gets set the every time the branch trace buffer wraps from PMD[15] to PMD[8]. Once set, the full bit remains set until explicitly cleared by software, i.e. it is a sticky bit. Software can reset the bbi index and the full bit by writing to PMD[16].

Figure 6-23. IA-64 Branch Trace Buffer Index Register Format (PMD[16])

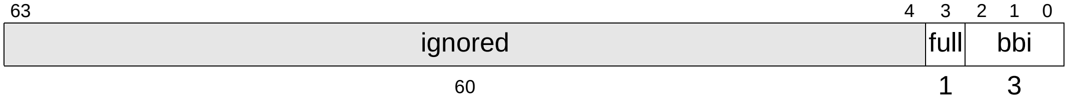


Table 6-24. IA-64 Branch Trace Buffer Index Register Fields (PMD[16])

Field	Bit Range	Description
bbi	2:0	Branch Buffer Index [Range 0..7 - Index 0 indicates PMD[8]] Pointer to the next branch trace buffer entry to be written. if full=1: points to the oldest recorded branch/target if full=0: points to the next location to be written
full	3	Full Bit (sticky) if full=1: branch trace buffer has wrapped if full=0: branch trace buffer has not wrapped

6.2.9 Processor Reset, PAL Calls, and Low Power State

Processor Reset: On processor hardware reset bits oi, ev of all PMC registers are zero, and PMV.m is set to one. This ensures that no interrupts are generated, and events are not externally visible. On reset, PAL firmware ensures that the instruction address range check, the opcode matcher and the data address range check are initialized as follows:

- PMC[13].ta=1,
- PMC[8,9].mifb=1111, PMC[8,9].mask{29:3}= “all 1s”, PMC[8,9].match{59:33}= “all 0s”, and
- PMC[11].pt is 1.

All other performance monitoring related state is undefined.

PAL Call: As defined in Chapter 11, "IA-64 Processor Abstraction Layer"in Volume 2, the PAL call PAL_PERF_MON_INFO provides software with information about the implemented performance monitors. The Itanium processor specific values are summarized in Table 6-25.

Low Power State: To ensure that monitor counts are preserved when the processor enters low power state, PAL_LIGHT_HALT freezes event monitoring prior to powering down the processor. PAL_LIGHT_HALT preserves the original value of the PMC[0] register.

Table 6-25. Information Returned by PAL_PERF_MON_INFO for the Itanium™ Processor

PAL_PERF_MON_INFO Return Value	Description	Itanium™ Processor-specific Value
PAL_RETIRED	8-bit unsigned event type for counting the number of untagged retired IA-64 instructions.	0x08
PAL_CYCLES	8-bit unsigned event type for counting the number of running CPU cycles.	0x12
PAL_WIDTH	8-bit unsigned number of implemented counter bits.	32
PAL_GENERIC_PM_PAIRS	8-bit unsigned number of generic PMC/PMD pairs.	4
PAL_PMCmask	256-bit mask defining which PMC registers are populated.	0x3FFF
PAL_PMDmask	256-bit mask defining which PMD registers are populated.	0x3FFFF
PAL_CYCLES_MASK	256-bit mask defining which PMC/PMD counters can count running CPU cycles (event defined by PAL_CYCLES)	0xF0
PAL_RETIRED_MASK	256-bit mask defining which PMC/PMD counters can count untagged retired IA-64 instructions (event defined by PAL_RETIRED)	0x10

6.2.10 References

- [gprof] S.L. Graham S.L., P.B. Kessler and M.K. McKusick, "gprof: A Call Graph Execution Profiler", Proceedings SIGPLAN'82 Symposium on Compiler Construction; SIGPLAN Notices; Vol. 17, No. 6, pp. 120-126, June 1982.
- [Lebeck] Alvin R. Lebeck and David A. Wood, "Cache Profiling and the SPEC benchmarks: A Case Study", Tech Report 1164, Computer Science Dept., University of Wisconsin - Madison, July 1993.
- [VTune] Mark Atkins and Ramesh Subramaniam, "PC Software Performance Tuning", IEEE Computer, Vol. 29, No. 8, pp. 47-54, August 1996.
- [WinNT] Russ Blake, "Optimizing Windows NT™", Volume 4 of the Microsoft "Windows NT Resource Kit for Windows NT Version 3.51", Microsoft Press, 1995.

Performance Monitor Events 7

This chapter summarizes the Itanium processor events and describes how to compute commonly used performance metrics from them. The event summaries are grouped as follows:

- Basic Events: clock cycles, retired instructions (Section 7.1).
- Instruction Execution: instruction decode, issue and execution, data and control speculation, and memory operations (Section 7.2).
- Cycle Accounting Events: stall cycle breakdowns (Section 7.3).
- Branch Events: branch prediction (Section 7.4).
- Memory Hierarchy: instruction prefetch, instruction and data caches (Section 7.5).
- System Events: operating system monitors, instruction and data TLBs (Section 7.6).

Note: The Itanium processor provides elaborate features to collect performance metrics to varying degrees of details. The user must have a good understanding of the architected performance monitor mechanisms before attempting to collect data (see Volume 2, Chapter 7 for more details). Also correct setup of the configuration register(s), privilege levels and other parameters are required for generating data that is both meaningful and correct.

The tables in the subsequent sections define events by specifying four attributes: symbolic event name, a brief event description, the PMC/PMD counter that can count the event, and a hexadecimal event code. Event codes and PMC/PMD counters are specified only for “monitored” events that are directly measurable by the processor. Some performance metrics are not directly measurable, but can be computed by combining or restricting one or more monitored event counts. These metrics are listed in the tables as “derived” events.

Events with no suffix are directly measured in hardware. Events with a “.a” suffix are also directly measured in hardware; the “.a” events are event name aliases. Events with a “.u” suffix are directly measured in hardware; however, they require an event code with a specific unit mask setting. PLease refer to Section 6.2.2, "Performance Counter Registers" for more details on umask. Events with a “.d” suffix are not measured in hardware directly, but can be computed from two or more measured events.

7.1 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.u) includes predicated true and false instructions, and `nops`, but excludes RSE operations. These instruction categories (and others) can be monitored as separate events (for details see Section 7.2). Table 7-2 defines IPC and average instructions/cycles per ISA transition metrics.

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

Execution Monitors	Description	PMC/PMD	Event Code / Umask
CPU_CYCLES	CPU Cycles	4,5,6,7	0x12
IA64_INST_RETIRED.u	Retired IA-64 Instructions	4,5	0x08 / 0x0
IA32_INSTR_RETIRED	IA-32 Instructions Retired	4,5,6,7	0x15
ISA_TRANSITIONS	IA-64 to IA-32 ISA Transitions	4,5,6,7	0x14

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.u / CPU_CYCLES[IA64]
IA-32 Instruction per Cycle	IA32_INSTR_RETIRED / CPU_CYCLES[IA32]
Average IA-64 Instructions/Transition	IA64_INST_RETIRED.u/ (ISA_TRANSITIONS*2)
Average IA-32 Instructions/Transition	IA32_INSTR_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)

7.2 Instruction Execution

This section describes events related to instruction issue and retirement (Table 7-3), multimedia and floating-point (Table 7-5), data and control speculation (Table 7-6), as well as memory instruction monitors (Table 7-9).

Table 7-3. Instruction Issue and Retirement Events

Decode, Issue, Retirement Monitors	Description	PMC/PMD	Event Code
INST_DISPERSED	Instructions Dispersed	4,5,6,7	0x2d
EXPL_STOPS	Explicit Stops	4,5,6,7	0x2e
IMPL_STOPS_DISPERSED	Implicit Stops	4,5,6,7	0x2f
IA64_TAGGED_INSTRS_RETIRED	Retired Tagged IA-64 Instructions	4,5	0x08[a]
NOPS_RETIRED	Retired NOP Instructions	4,5	0x30
PREDICATE_SQUASHED_RETIRED	Instructions Squashed Due to Predicate Off	4,5	0x31
RSE_REFERENCES_RETIRED	RSE Accesses	4,5,6,7	0x65
RSE_LOADS_RETIRED	RSE Load Accesses	4,5,6,7	0x32
RSE_STORES_RETIRED.d	RSE Store Accesses	None	Derived[b]

a. See Section Table 7-4., "Retired Event Selection by Opcode Match" for umask values.
b. RSE_STORES_RETIRED.d = (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) depends on the stop bits in the instruction stream (EXPL_STOPS) as well as functional unit availability (IMPL_STOPS_DISPERSED).

Retired instruction counts (IA64_TAGGED_INSTRS_RETIRED, NOPS_RETIRED) are based on tag information specified by the address range check and opcode match facilities described in Section 6.1.3, "Event Qualification". The tagged retired instruction counts include predicated off instructions but exclude RSE operations. 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.u event can be used (PMC/PMD$_4$ only), or the IA64_TAGGED_INSTRS_RETIRED event. IA64_TAGGED_INSTRS_RETIRED counts number of retired instructions (includes predicated off instructions) that match the instruction address range and opcode match settings in the IBR and PMC registers. The TAG_SELECT unit mask defined in Table 7-4 always qualifies the event count of IA64_TAGGED_INSTRS_RETIRED with either the opcode match register PMC$_8$ or PMC$_9$. Note that the setting of PMC$_8$ qualifies all downstream event monitors (see Section 6.1.3, "Event Qualification" for details). To ensure that other monitored events are counted independent of the opcode matcher, m, i, f, b bits and all mask bits of PMC$_8$ (Table 7-24) should be set to one (all opcodes match). The settings of PMC$_9$ do not affect other event monitors.

Table 7-4. 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

The floating-point monitors listed in Table 7-5 capture dynamic run-time information (FP_FLUSH_TO_ZERO, FP_SIR_FLUSH).

Table 7-5. Floating-point Execution Monitors

Floating-point Monitors	Description	PMC/PMD	Event Code
FP_FLUSH_TO_ZERO	FP Result Flushed to Zero	4,5,6,7	0x0b
FP_SIR_FLUSH	FP SIR Flush Cycles	4,5,6,7	0x0c

As described in Table 7-6, monitors for control and data speculation capture dynamic run-time information: the number of failed `chk.s` instructions (INST_FAILED_CHKS_RETIRED), the number of advanced check loads and check loads (ALAT_INST_CHKA_LDC) and failed advanced check loads and no-clear check loads (ALAT_INST_FAILED_CHKA_LDC) as seen by the ALAT. The number of retired `chk.s` instructions is monitored by the IA64_TAGGED_INSTRS_RETIRED event with the appropriate opcode mask. Since the Itanium processor ALAT is updated by operations on mispredicted branch paths the number of advanced check loads needs an explicit event (ALAT_INST_CHKA_LDC). Finally, the ALAT_CAPACITY_MISS event can be used to monitors ALAT overflows.

Table 7-6. Control and Data Speculation Monitors

Control and Data Speculation Monitors	Description	PMC/PMD	Event Code
INST_FAILED_CHKS_RETIRED	Failed Speculative Check Loads	4,5,6,7	0x35
ALAT_INST_CHKA_LDC	Advanced Check Loads	4,5,6,7	0x36
ALAT_INST_FAILED_CHKA_LDC	Failed Advanced Check Loads	4,5,6,7	0x37
ALAT_CAPACITY_MISS	ALAT Entry Replaced	4,5,6,7	0x38

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

Table 7-7. INST_TYPE Unit Mask for Control and Data Speculation Events

Speculative/Advanced INST_TYPE	PMC.umask {19:16}	Description
NONE	xx00	no instructions are counted
INTEGER	xx01	count speculative/advanced integer instructions only
FP	xx10	count speculative/advanced floating-point instructions only
ALL	xx11	count both integer and floating-point speculative/advanced instructions

Table 7-8. Itanium™ Processor Control/Data Speculation Performance Metrics

Performance Metric	Performance Monitor Equation
Control Speculation Miss Ratio	INST_FAILED_CHKS_RETIRED / IA64_TAGGED_INSTRS_RETIRED[chk.s]
Data Speculation Miss Ratio	ALAT_INST_FAILED_CHKA_LDC / ALAT_INST_CHKA_LDC
ALAT Capacity Miss Ratio	ALAT_CAPACITY_MISS/ IA64_TAGGED_INSTRS_RETIRED[ld.sa,ld.a,ldfp.a,ldfp.sa]

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. Itanium™ Processor Memory Events

Memory Monitors	Description	PMC/PMD	Event Code
LOADS_RETIRED	Retired Loads	4,5,6,7	0x6c
STORES_RETIRED	Retired Stores	4,5,6,7	0x6d
UC_LOADS_RETIRED	Retired Uncacheable Loads	4,5,6,7	0x6e
UC_STORES_RETIRED	Retired Uncacheable Stores	4,5,6,7	0x6f
MISALIGNED_LOADS_RETIRED	Retired Misaligned Load Instructions	4,5,6,7	0x70
MISALIGNED_STORES_RETIRED	Retired Misaligned Store Instructions	4,5,6,7	0x71

7.3 Cycle Accounting Events

As described in Section 6.1.1.4, "Cycle Accounting", the Itanium processor provides eight directly measured and four derived stall cycle monitors. Table 7-10 lists the Itanium processor stall events.

Table 7-10. Itanium™ Processor Stall Cycle Monitors

Stall Accounting Monitors	Description	PMC/PMD	Event Code
BRANCH_MISPRED_CYCLE	Branch Mispredict Stall Cycle	4,5,6,7	0x00
DATA_ACCESS_CYCLE	Data Access Stall Cycle	4,5,6,7	0x03
EXEC_LATENCY_CYCLE	Execution Latency Stall Cycle	4,5,6,7	0x02
INST_ACCESS_CYCLE	Instruction Access Cycle	4,5,6,7	0x01
BRANCH_CYCLE	Combined Branch Stall Cycle	4,5,6,7	0x04
MEMORY_CYCLE	Combined Memory Stall Cycle	4,5,6,7	0x07
EXECUTION_CYCLE	Combined Execution Stall Cycle	4,5,6,7	0x06
INST_FETCH_CYCLE	Combined Instruction Fetch Stall Cycle	4,5,6,7	0x05
RSE_ACTIVE_CYCLE.d	RSE Active Cycle	4,5,6,7	Derived[a]
ISSUE_LIMIT_CYCLE.d	Issue Limit Cycle	4,5,6,7	Derived[b]
TAKEN_BRANCH_CYCLE.d	Taken Branch Cycle	4,5,6,7	Derived[c]
FETCH_WINDOW_CYCLE.d	Fetch Window Cycle	4,5,6,7	Derived[d]

[a] RSE_ACTIVE_CYCLE.d = (MEMORY_CYCLE) –(DATA_ACCESS_CYCLE).
[b] ISSUE_LIMIT_CYCLE.d = (EXECUTION_CYCLE) –(EXEC_LATENCY_CYCLE).
[c] TAKEN_BRANCH_CYCLE.d = (BRANCH_CYCLE) –(BRANCH_MISPRED_CYCLE).
[d] FETCH_WINDOW_CYCLE.d = (INST_FETCH_CYCLE) –(INST_ACCESS_CYCLE).

7.4 Branch Events

The five measured Itanium processor branch events listed in Table 7-11 expand into over fifty measurable branch metrics by using the unit masks described on the event pages. BR_PATH_PREDICTION counts branches based on branch direction (taken/not taken) and prediction outcome (mispredict or not). BR_MISPREDICT_DETAIL and BR_MWAY_DETAIL provide finer resolution, and break down branch events by mispredict reasons (correctly predicted, wrong branch outcome, wrong target) and by the Itanium processor branch prediction structures. BR_TAKEN_DETAIL counts taken branches on per instruction slot basis, and, in conjunction with the instruction address range check, can be used for detailed branch profiling. BRANCH_EVENT counts the number of events captured in the branch trace buffer.

Table 7-11. Itanium™ Processor Branch Monitors

Branch Events	Description	PMC/PMD	Event Code
BR_PATH_PREDICTION	Branch Path Prediction	4,5,6,7	0x0f[a]
BR_MISPREDICT_DETAIL	Branch Mispredict Detail	4,5,6,7	0x10[a]
BR_MWAY_DETAIL	Multiway Branch Detail	4,5,6,7	0x0e[a]
BR_TAKEN_DETAIL	Taken Branch Detail	4,5,6,7	0x0d[a]

[a] See following sections for more umask values.

All branch events can be qualified by instruction address range and opcode matching as described in Section 6.1.3, “Event Qualification” on page 6-8. 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 0 or 1)
    and (branch in slot 1 is taken)
    and (branch in slot 0 is NOT taken)
or  ((address range and opcode match on instruction slot 0 or 1 or 2)
    and (branch in slot 1 is NOT taken)
    and (branch in slot 0 is NOT taken)
)
```

7.4.1 BR_PATH_PREDICTION

One event unit mask (BRANCH_PATH_RESULT) allows branch monitoring to be constrained to combinations of taken/not taken.

Table 7-12. Branch Selection Based on Branch Prediction Result and Branch Direction

BRANCH_PATH_RESULT	PMC.umask {19:16}	Description
MISPRED_NT	0000	Incorrectly predicted path and Not taken branches.
MISPRED_TAKEN	0001	Incorrectly predicted path and taken branches.
OKPRED_NT	0010	Correctly predicted path and Not taken branches.
OKPRED_TAKEN	0011	Correctly predicted path and taken branches.

7.4.2 BR_MISPREDICT_DETAIL

BR_MISPREDICT_DETAIL can categorize branch mispredictions by mispredict reason (correctly predicted, wrong path or wrong target). Below event unit mask (PREDICTION_RESULT) allows branch monitoring to be constrained to combination of prediction results.

Table 7-13. Branch Selection Based on Branch Prediction Outcome

PREDICTION_RESULT	PMC.umask {19:16}	Description
ALL_PREDICTIONS	0000	count branches without regard to prediction result
CORRECT_PREDICTION	0001	count correctly predicted branches only. For taken branches this means that both the path and the target prediction are correct. For not taken branches, only the path prediction was correct
WRONG_PATH	0010	count mispredicted branches due to wrong branch path only (taken or not taken branches)
WRONG_TARGET	0011	count mispredicted branches due to wrong target only (only happens for taken branches whose path was predicted correctly)

7.4.3 BR_MWAY_DETAIL

BR_MWAY_DETAIL monitors the outcome of multiway branches, i.e. any MBB or BBB bundles with at least one branch. Two event unit masks (BRANCH_PATH and PREDICTION_RESULT) allow branch monitoring to be constrained to combinations of taken/not taken (Table 7-14) and branch prediction outcomes (Table 7-15).

Table 7-14. Branch Selection Based on Branch Prediction Outcome

PREDICTION_RESULT	PMC.umask {19:16}	Description
ALL_PREDICTIONS	xx00	count branches without regard to prediction result
CORRECT_PREDICTION	xx01	count correctly predicted branches only. For taken branches this means that both the path and the target prediction are correct. For not taken branches, only the path prediction was correct
WRONG_PATH	xx10	count mispredicted branches due to wrong branch path only (taken or not taken branches)
WRONG_TARGET	xx11	count mispredicted branches due to wrong target only (only happens for taken branches whose path was predicted correctly)

Table 7-15. Multi-way Branch Selection Based on Branch Path

BRANCH_PATH	PMC.umask {19:16}	Description
NOT_TAKEN	10xx	count not-taken branches only
TAKEN	11xx	count taken branches only
ALL_PATHS	0xxx	counts all branches (taken or not-taken)

7.4.4 BR_TAKEN_DETAIL

BR_TAKEN_DETAIL monitors taken branches based on their instruction slot number. The SLOT_MASK unit mask defined in Table 7-16 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.

Table 7-16. Slot Unit Mask for BR_TAKEN_DETAIL

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

7.5 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 7.5.1)
- L1 Data Cache (Section 7.5.2)
- L2 Unified Cache (Section 7.5.3)
- L3 Cache (Section 7.5.4)

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 7-17.

7.5.1 L1 Instruction Cache and Prefetch

Table 7-18 summarizes the eight events that the Itanium processor provides to monitor the L1 instruction cache and prefetch activity. The instruction fetch monitors distinguish between demand fetch (L1I_READS, L1I_MISSES) and prefetch activity (L1I_IPREFETCHES, L2_INST_PREFETCHES). 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 TLB 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 matching facilities described in Section 6.1.3, "Event Qualification" on page 6-8. 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])" on page 6-18 for details).

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

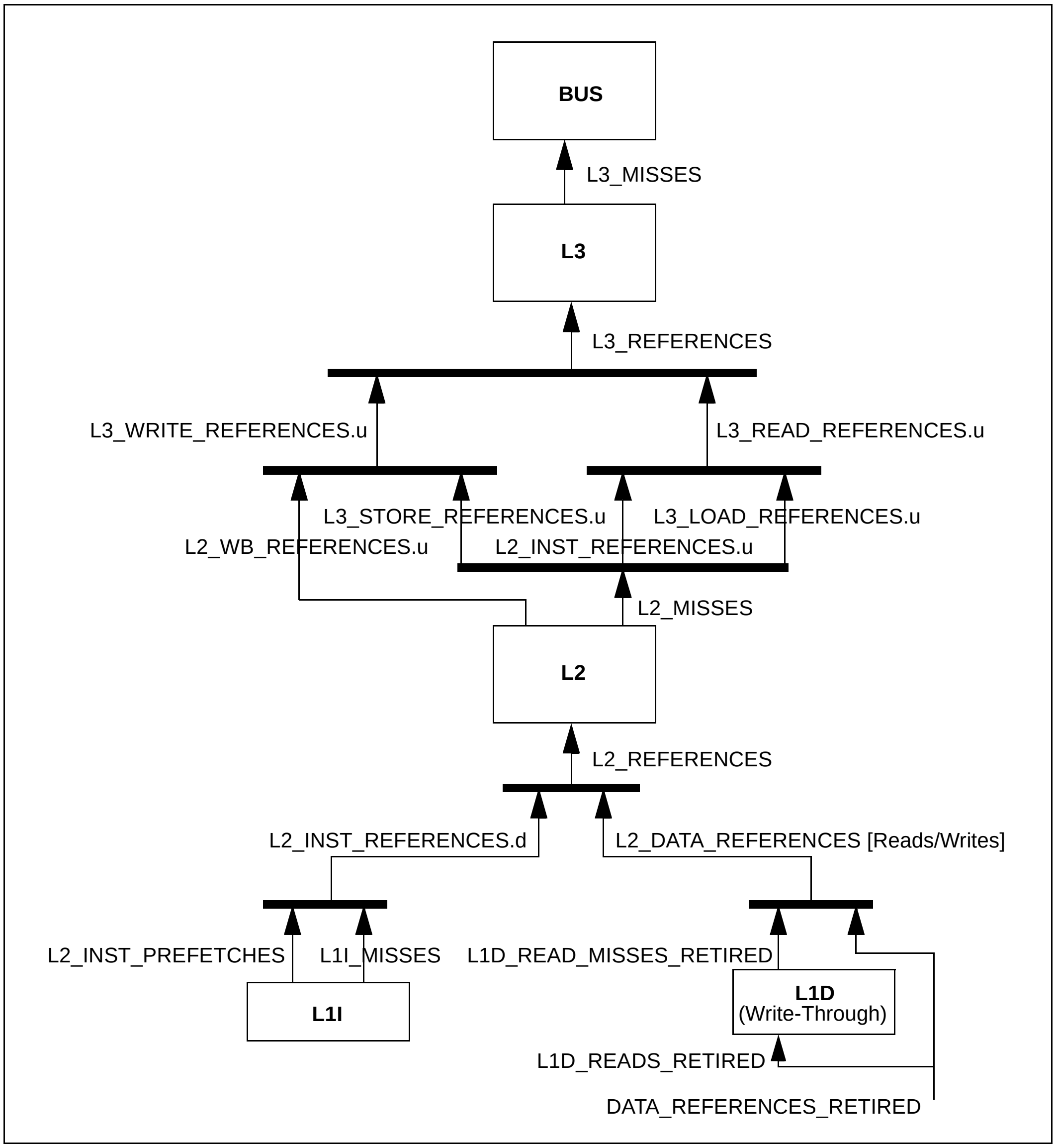


Table 7-17. Itanium™ Processor Cache Performance Ratios

Performance Metric	Itanium Processor Performance Monitor Equation
L1I Miss Ratio	L1I_MISSES / L1I_REFERENCES.d
L1D Read Miss Ratio	L1D_READ_MISSES_RETIRED / L1D_READS_RETIRED
L2 Miss Ratio	L2_MISSES / L2_REFERENCES
L2 Data Miss Ratio	L3_DATA_REFERENCES.d / L2_DATA_REFERENCES
L2 Instruction Miss Ratio (includes prefetches)	L3_INST_REFERENCES.u/L2_INST_REFERENCES.d
L2 Data Read Miss Ratio	L3_LOAD_REFERENCES.u / L2_DATA_READS.u
L2 Data Write Miss Ratio	L3_STORE_REFERENCES.u / L2_DATA_WRITES.u

Table 7-17. Itanium™ Processor Cache Performance Ratios (Continued)

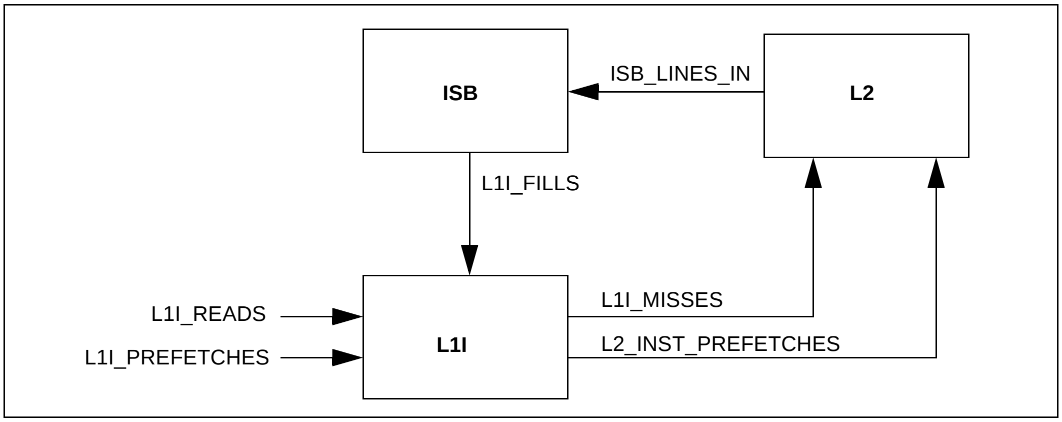
Performance Metric	Itanium Processor Performance Monitor Equation
L2 Instruction Fetch Ratio	L1I_MISSES / L2_REFERENCES
L2 Data Ratio	L2_DATA_REFERENCES / L2_REFERENCES
L3 Miss Ratio	L3_MISSES / L2_MISSES
L3 Data Miss Ratio	(L3_LOAD_MISSES.u + L3_STORE_MISSES.u) / L3_DATA_REFERENCES.d
L3 Instruction Miss Ratio	L3_INST_MISSES.u / L3_INST_REFERENCES.u
L3 Data Read Ratio	L3_LOAD_REFERENCES.u / L3_DATA_REFERENCES.d
L3 Data Ratio	L3_DATA_REFERENCES.d / L3_REFERENCES

Table 7-18. L1 Instruction Cache and Instruction Prefetch Monitors

	Description	PMC/PMD	Event Code
L1I Monitors			
L1I_REFERENCES.d	L1 Instruction Cache References	None	Derived[a]
L1I_READS	L1 Instruction Cache Reads	4,5,6,7	0x20
L1I_FILLS	L1 Instruction Cache Fills	4,5,6,7	0x21
L1I_MISSES	L1 Instruction Cache Misses	4,5,6,7	0x22
INSTRUCTION_EAR_EVENTS	Instruction EAR Events	4,5,6,7	0x23
I-Prefetch Monitors			
L1I_IPREFETCHES	L1 Instruction Prefetch Requests	4,5,6,7	0x24
L2_INST_PREFETCHES	L2 Instruction Prefetch Requests	4,5,6,7	0x25
ISB_LINES_IN	Instruction Streaming Buffer Lines In	4,5,6,7	0x26

[a] L1I_REFERENCES.d =(L1I_READS) +(L1I_IPREFETCHES).

Figure 7-2. L1 Instruction Cache and Prefetch Monitors



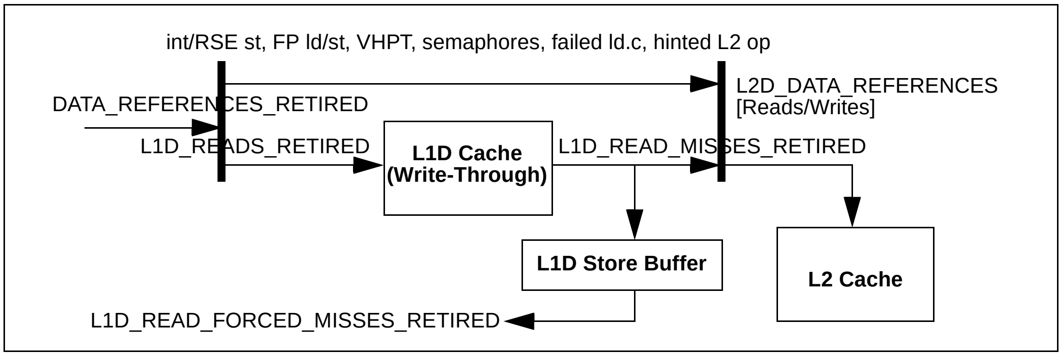
7.5.2 L1 Data Cache

Table 7-19 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, floating-point memory operations, VHPT references, semaphores, check loads and hinted L2 memory references are serviced by the L2 cache. DATA_REFERENCES_RETIRED is the number of issued data memory references. The count includes wrong-path operations. 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) is the number of data requests prior to cache line merging, and can be broken down into reads and writes. The DATA_EAR_EVENTS monitor (not shown in Figure 7-3) counts how many data cache or TLB 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 7-19. L1 Data Cache Monitors

L1D Monitors	Description	PMC/PMD	Event Code / Umask
DATA_REFERENCES_RETIRED	Retired Data Memory References	4,5,6,7	0x63
L1D_READS_RETIRED	L1 Data Cache Reads	4,5,6,7	0x64
L1D_READ_MISSES_RETIRED	L1 Data Cache Read Misses	4,5,6,7	0x66
L1D_WAY_MISPREDICT.u	L1 Data Cache Way Mispredicts	4,5,6,7	0x33 / 0x2
L1D_READ_FORCED_MISSES_RETIRED	L1 Data Cache Forced Load Misses	4,5,6,7	0x6b
L2_DATA_REFERENCES	L2 Data References	4,5,6,7	0x69
DATA_EAR_EVENTS	L1 Data Cache EAR Events	4,5,6,7	0x67

Figure 7-3. L1 Data Cache Monitors



7.5.3 L2 Unified Cache

Table 7-20 lists 9 measured and 2 derived events that monitor the Itanium processor L2 cache. Refer to Figure 7-1 for graphical view of the L2 cache monitors.

Table 7-20. L2 Cache Monitors

L2 Monitors	Description	PMC/PMD	Event Code / Umask
L2_REFERENCES	L2 References	4,5,6,7	0x68
L2_INST_REFERENCES.d	L2 Instruction References	None	Derived[a]
L2_INST_FETCHES.a	L2 Instruction Fetches	None	Alias[b]
L2_INST_PREFETCHES	L2 Instruction Prefetch Requests	4,5,6,7	0x25
L2_DATA_REFERENCES	L2 Data References	4,5,6,7	0x69
L2_DATA_READS.u	L2 Data Reads	None	0x69 / 0x1
L2_DATA_WRITES.u	L2 Data Writes	None	0x69 / 0x2
L2_MISSES	L2 Misses	4,5,6,7	0x6a
L2_FLUSHES	L2 Flushes	4,5,6,7	0x76
L2_FLUSH_DETAILS	L2 Flush Details	4,5,6,7	0x77

a. L2_INST_REFERENCES.d = (L1I_MISSES) +(L2_INST_PREFETCHES).
b. This is equal to L1I_MISSES.

L2_REFERENCES, L2_INST_PREFETCHES and L2_DATA_REFERENCES are counted in terms of number of requests seen by the L2. L2_MISSES are counted in terms of the number of L2 cache line requests sent to the L3. L2_FLUSHES and L2_FLUSH_DETAILS count and break-down the number of L2 flushes due to address and bank conflicts.

L1D_READ_FORCED_MISSES_RETIRED counts the number of loads that were bypassed from an earlier store.

7.5.4 L3 Cache

Table 7-21 lists 23 L3 cache measured events and one derived events. Using unit masks, two events (L3_READS, L3_WRITES) can be specialized (hit/miss/all accesses, instruction/data/all references) to count a number of derived L3 events. Refer to the event pages for L3_READS or L3_WRITES for details on L3 unit mask usage. Refer to Figure 7-1 for graphical view of the L3 cache monitors.

Table 7-21. L3 Cache Monitors

L3 Monitors	Description	PMC/PMD	Event Code
L3_REFERENCES	L3 References	4,5,6,7	0x7b
L3_MISSES	L3 Misses	4,5,6,7	0x7c
L3_READS	L3 Reads	4,5,6,7	0x7d
L3_WRITES	L3 Writes	4,5,6,7	0x7e
L3_LINES_REPLACED	L3 Cache Lines Replaced	4,5,6,7	0x7f

Table 7-21. L3 Cache Monitors (Continued)

L3 Monitors	Description	PMC/PMD	Event Code
L3_INST_REFERENCES.u	L3 Instruction References	4,5,6,7	Umask[a]
L3_INST_MISSES.u	L3 Instruction Fetch Misses	4,5,6,7	Umask[a]
L3_INST_HITS.u	L3 Instruction Fetch Hits	4,5,6,7	Umask[a]
L3_DATA_REFERENCES.d	L3 Data References	4,5,6,7	Derived
L3_LOAD_REFERENCES.u	L3 Load References	4,5,6,7	Umask[a]
L3_LOAD_MISSES.u	L3 Load Misses	4,5,6,7	Umask[a]
L3_LOAD_HITS.u	L3 Load Hits	4,5,6,7	Umask[a]
L3_READ_REFERENCES.u	L3 Read References	4,5,6,7	Umask[a]
L3_READ_MISSES.u	L3 Read Misses	4,5,6,7	Umask[a]
L3_READ_HITS.u	L3 Read Hits	4,5,6,7	Umask[a]
L3_STORE_REFERENCES.u	L3 Store References	4,5,6,7	Umask[b]
L3_STORE_MISSES.u	L3 Store Misses	4,5,6,7	Umask[b]
L3_STORE_HITS.u	L3 Store Hits	4,5,6,7	Umask[b]
L2_WB_REFERENCES.u	L2 Write Back References	4,5,6,7	Umask[b]
L2_WB_MISSES.u	L2 Write Back Misses	4,5,6,7	Umask[b]
L2_WB_HITS.u	L2 Write Back Hits	4,5,6,7	Umask[b]
L3_WRITE_REFERENCES.u	L3 Write References	4,5,6,7	Umask[b]
L3_WRITE_MISSES.u	L3 Write Misses	4,5,6,7	Umask[b]
L3_WRITE_HITS.u	L3 Write Hits	4,5,6,7	Umask[b]

[a] Refer to Table 7-22 for umask values.
[b] Refer to Table 7-23 for umask values.

Table 7-22. L3_READS Derived Events

L3_Reads	PMC.umask{19:18}		
PMC.umask{17:16}	INSTR_FETCH (01)	DATA_READ (10)	ALL_READS (11)
HIT (01)	L2_INST_HITS.u	L2_LOAD_HITS.u	L2_READ_HITS.u
MISS (10)	L2_INST_MISSES.u	L2_LOAD_MISSES.u	L2_READ_MISSES.u
ALL (11)	L2_INST_REFERENCES.u	L2_LOAD_REFERENCES.u	L2_READ_REFERENCES.u

Table 7-23. L3_WRITES Derived Events

L3_WRITES	PMC.umask[19:18]		
PMC.umask{17:16}	DATA_WRITE (01)	L1_WRITE_BACK (10)	ALL_WRITES (11)
HIT (01)	L2_STORE_HITS.u	L1_WB_HITS.u	L2_WRITE_HITS.u
MISS (10)	L2_STORE_MISSES.u	L1_WB_MISSES.u	L2_WRITE_MISSES.u
ALL (11)	L2_STORE_REFERENCES.u	L1_WB_REFERENCES.u	L2_WRITE_REFERENCES.u

7.6 System Events

Table 7-24 defines seven measured and one derived system monitor. 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.a) 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. Two events (EXTERN_BPM_PINS_0_TO_3 and EXTERN_BPM_PINS_4_TO_5) are provided to monitor external platform events.

Table 7-24. Itanium™ Processor System Monitors

System Monitors	Description	PMC/PMD	Event Code
CODE_DEBUG_REGISTER_MATCHES.a	Code Debug Register Matches	None	Derived[a]
DATA_DEBUG_REGISTER_MATCHES.d	Data Debug Register Matches	None	Derived[b]
PIPELINE_FLUSH	Pipeline Flush	4,5,6,7	0x33
CPU_CPL_CHANGES	Privilege Level Changes	4,5,6,7	0x34
CPU_CYCLES	CPU Cycles	4,5,6,7	0x12
EXTERN_BPM_PINS_0_TO_3	Counts the number of times external BPM pins 0 through 23 were asserted	4,5,6,7	0x5e
EXTERN_BPM_PINS_4_TO_5	Counts the number of times external BPM pins 4 and 5 were asserted	4,5,6,7	0x5f

[a] CODE_DEBUG_REGISTER_MATCHES.a = IA64_TAGGED_INSTRS_RETIRED.

[b] DATA_DEBUG_REGISTER_MATCHES.d = LOADS_RETIRED + STORES_RETIRED.

Table 7-25 lists the TLB performance metrics that can be computed using these 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. Figure 7-4 gives a graphical summary.

Table 7-25. Itanium™ Processor TLB Performance Metrics

Performance Metric	Performance Monitor Equation
ITLB Miss Ratio	ITLB_MISSES_FETCH / L1I_READS
DTLB Miss Ratio	DTLB_MISSES / DATA_REFERENCES_RETIRED
DTC Miss Ratio	DTC_MISSES / DATA_REFERENCES_RETIRED

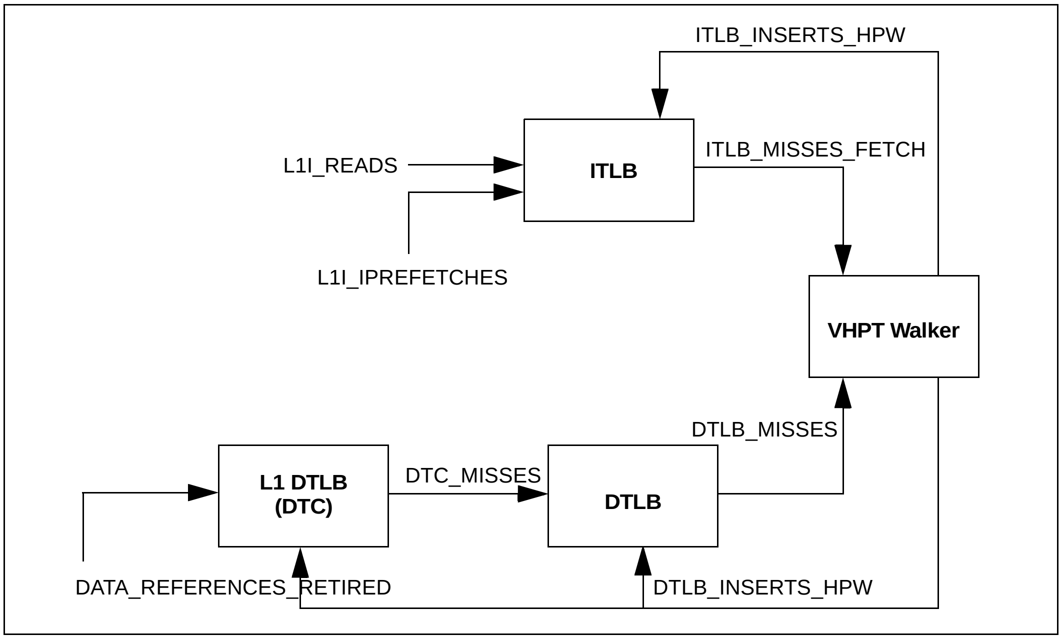
ITLB_REFERENCES.a and DTLB_REFERENCES.a are derived from the respective instruction/data cache access events. ITLB_MISSES_FETCH and DTLB_MISSES count TLB misses. ITLB_INSERTS_HPW and DTLB_INSERTS_HPW count the number of instruction/data TLB inserts performed by the Virtual Hash Page Table walker. The Itanium processor data TLB is a two level TLB; DTC_MISSES counts the number of first level data TLB misses.

Table 7-26. Itanium™ Processor Instruction and Data TLB Monitors

Instruction and Data TLB Monitors	Description	PMC/PMD	Event Code
ITLB_REFERENCES.a	Instruction Translation Buffer References	None	Derived[a]
ITLB_MISSES_FETCH	Instruction Translation Buffer Misses Demand Fetch	4,5,6,7	0x27
ITLB_EAR_EVENT.a	Instruction Translation Buffer EAR Event	None	Derived[b]
ITLB_INSERTS_HPW	ITLB Hardware Page Walker Inserts	4,5,6,7	0x28
DTLB_REFERENCES.a	DTLB References	4,5,6,7	Derived[c]
DTC_MISSES	DTC Misses	4,5,6,7	0x60
DTLB_MISSES	DTLB Misses	4,5,6,7	0x61
DTLB_EAR_EVENT.a	DTLB EAR Event	4,5,6,7	Derived[d]
DTLB_INSERTS_HPW	Hardware Page Walker Installs to DTLB	4,5,6,7	0x62

a. This is equal to L1I_READS.
b. This is equal to INSTRUCTION_EAR_EVENTS.
c. This is equal to DATA_REFERENCES_RETIRED.
d. This is equal to DATA_EAR_EVENTS.

Figure 7-4. Itanium™ Processor Instruction and Data TLB Monitors



Model Specific Behavior for IA-32 Instruction Execution

8

The Itanium processor is capable of executing IA-32 instructions in the IA-32 system environment (legacy lA-32 operating systems) provided the required platform and firmware support exists in the system. The Itanium processor is also capable of executing IA-32 instructions in the IA-64 system environment (IA-64 operating system). IA-64 operating system support for the capability of running IA-32 applications is defined by the respective operating system vendor. For more details on IA-32 instruction execution on IA-64 OS, please refer to Volume 1, Chapter 6 and Volume 2, Chapter 10.

Note that while Itanium processor supports execution of IA-32 applications, best performance and capabilities will be realized by using 64-bit optimized OSes and applications

In general, the behavior of IA-32 instructions on the Itanium processor is similar to that of the Pentium III processor except where noted. The following sections describe some of the key differences in behavior between IA-32 instruction execution on an Itanium processor and on the Pentium III processor. These differences do not prevent IA-32 legacy operating systems or IA-32 applications from operating correctly.

8.1 Processor Reset and Initialization

When RESET# is asserted, all IA-64 processors boot at a different reset location than IA-32 processors and start executing IA-64 64-bit code instead of IA-32 16-bit Real Mode code. Unlike IA-32 processors, IA-64 processors execute PAL firmware to test and initialize the processor and then continue execution in the IA-64 instruction set to boot the system. SAL firmware code can switch to the IA-32 instruction set as needed to execute IA-32 BIOS code. For more details on IA-64 processor reset, please refer to Chapter 11 and Chapter 24 of Volume 2.

8.2 New JMPE Instruction

A new IA-32 instruction JMPE has been defined for IA-64 processors. This instruction comes in two forms with an opcode for each. These opcodes will cause an Invalid Opcode fault on all IA-32 processors. For more details, refer to Chapter 5 of Volume 3.

8.3 System Management Mode (SMM)

SMM is superseded by the IA-64 Platform Management definition. This mechanism is designed to provide platform level interrupt support for both IA-32 and IA-64 operating systems. Please refer to Chapter 11 of Volume 2 for more details on PMI.

The IA-32 SMM and I/O Port Restart feature is not supported on the Itanium processor. Dynamically, powering off/on I/O devices on an I/O Port reference via system logic is not possible for IA-32 Operating Systems or IA-64 Operating Systems using the IA-32 SMM I/O Restart mechanism. I/O Restart has not been extended on IA-64 processors to intercept I/O Port references from the IA-64 instruction set via normal loads and stores on IA-64 processors.

Execution of the IA-32 RSM (Resume from SMM) instruction results an Invalid Opcode fault on all IA-64 processors.

8.4 Machine Check Abort (MCA)

The Itanium processor supports Pentium processor level machine checks in the IA-32 System Environment.

8.5 Model Specific Registers

The complete set of Model Specific Registers (MSRs) found on the Pentium III processor is not supported on the Itanium processor. For example, Model Specific Debug registers, Model Specific Test registers, Machine Check registers, and Model Specific Configuration registers are not supported.

Model Specific registers that are common to the Itanium processor and Pentium III processor use the Pentium III processor's bit definition and register assignment. The ITC, APIC_Base, MTRR and MAP registers are supported on the Itanium processor.

8.6 Cache Modes

Pentium processor and Pentium III processor SRAM Cache Mode is not supported on the Itanium processor.

SRAM is typically used on IA-32 processors to provide scratch RAM areas while running IA-32 boot and machine check code before memory is available. Both of these functions are now provided by IA-64 firmware while running IA-32 and IA-64 operating systems.

8.7 10-byte Floating-point Operand Reads and Writes

Many IA-32 FP instructions read and write 10 bytes to memory. Consider the case of 16-bit segment, where the read or write starts at offset 0xFFF8. Pentium III processor reads or writes 8 bytes then re-evaluates the linear address before reading or writing the final 2 bytes. Eight bytes are accessed at 0xfff8, and 2 bytes are accessed at 0x0000.

The Itanium processor evaluates the address once, then accesses all 10 bytes. Therefore, bytes 0xfff8 to 0x10001 will be accessed.

On a 10-byte operand read or write access, potential page faults and GP faults will return slightly different faulting addresses (linear addresses may wrap differently).

8.8 Floating-point Data Segment State

The Itanium processor reports a different value of the floating-point data segment state (FDS) after the execution of “FNOP” instruction (or any FP instruction that does not perform a memory reference). The contents of the data register are undefined if the prior non-control instruction did not have a memory operand. The Pentium III processor behaves as follows:

1. A FP non-transparent instruction which references memory will put the selector of the data segment used in the memory reference into FDS.
2. A FP non-transparent instruction which doesn’t reference memory will put the selector of SS into FDS and 0 into FEA.

If a segment override prefix is present on an instruction of the type specified in case 2, the overriding segment selector will be put into FDS instead of the selector of SS.

The Itanium processor behavior covers only case #1 described above. Note that this difference does not affect the running of IA-32 applications.

8.9 Writes to Reserved Bits during FXSAVE

During FXSAVE, the Itanium processor does not write any reserved bits, while the Pentium III processor may write reserved bits. The Itanium processor does one 10 byte access to save each FP register, whereas the Pentium III processor will do two 8 byte accesses causing writes to upper reserved bits.

8.10 Setting the Access/Dirty (A/D) Bit on Accesses that Cross a Page Boundary

In the IA-32 system environment, the Itanium processor sets a page's A/D bit even if a memory reference crosses a page boundary and the other page has a fault. This behavior is different from Pentium III processors which do not modify the A/D bit under the above conditions.

The above difference does not come into play in the IA-64 system environment.

8.11 Enhanced Floating-point Instruction Accuracy

On the Itanium processor, FP transcendental instructions will return more accurate (hence slightly different) answers than Pentium III processor. This behavior falls into 3 categories:

- **F2XM1, FYL2X, FYL2XP1, FPATAN Instructions**
  More accurate algorithms will result in answers which may differ from Pentium III processor by 1 **u**nit in the **l**ast **p**lace (ulp). Also, for FYL2X and FYL2XP1, when x or x+1 respectively is a power of two, the Precision exception is not signaled (since log(2^k) where, k is integral, is exact).

- **FPTAN, FSIN, FCOS, FSINCOS Instructions**
  New algorithms on Itanium processor include a more accurate argument reduction scheme. Although more accurate, the algorithms implemented on Itanium processor can produce answers which are different from those returned on Pentium III processor.
- **FPREM, FPREM1 Instructions**
  No change.

8.12 RCPSS, RCPPS, RSQRTSS, RSQRTPS Instruction Differences

These four instructions are single and parallel approximations of divide and square root operations. The Itanium processor will calculate these functions to a higher accuracy than previous implementations. resulting in different answers. 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 $1.5*10^{-16}$.

8.13 Read/Write Access Ordering

In general, the order of reads/writes within any complex IA-32 instruction is model specific even among IA-32 processors. Different Intel processors have different access ordering behavior; for example, internal operation ordering varies between the 80486, Pentium, Pentium III and Itanium processors.

8.14 Multiple IOAPIC Redirection Table Entries

If multiple IOAPIC Redirection Table Entries (RTE) share the same vector, and at least one RTE is programmed as logical delivery mode in which the selected local APIC destinations overlap with the other RTEs with the same vector, some of the selected local APICs might not receive the interrupt when the pins that correspond to these RTEs are asserted.

8.15 Self Modifying Code (SMC)

The Itanium processor provides the same SMC support as the Pentium processor. Also, a branch instruction is required between the store that modifies instruction(s) and the modified code.

8.16 Raising an Alignment Check (AC) Fault

The Pentium III processor checks and raises AC fault before a page fault. The Itanium processor checks and raise a page fault before an AC fault.

8.17 Maximum Number of IA-64 Processors Supported in MP System Running Legacy IA-32 OS (IA-32 system environment)

Similar to the case of IA-32 processors in an MP system, the maximum number of IA-64 processors supported in a MP system running legacy IA-32 OS (IA-32 system environment) is 16. However, in MP systems with IA-32 processors, the number of IA-32 processors can be extended beyond 16 with additional platform enhancements while the limit for the number of IA-64 processors running IA-32 OS in a MP system is limited to 16.