Dual-Core Update to the Intel® Itanium® 2 Processor Reference Manual

For Software Development and Optimization

Revision 0.9

January 2006

Document Number: 308065-001

Notice: This document contains information on products in the design phase of development. The information here is subject to change without notice. Do not finalize a design with this information.

Contents

Figures

Tables

Revision History

Document Number	Revision Number	Description	Date
308065-001	0.9	• Initial release of the document.	January 2006

1 Introduction

This document is an update to the *Intel® Itanium® 2 Processor Reference Manual for Software Development and Optimization*. This update is meant to give guidance on the changes that the dual-core Intel® Itanium® 2 processor, code named Montecito, brings to the existing Itanium 2 processor family.

1.1 Terminology

The following definitions are for terms that will be used throughout this document:

Term	Definition
Dispersal	The process of mapping instructions within bundles to functional units
Bundle rotation	The process of bringing new bundles into the two-bundle issue window
Split issue	Instruction execution when an instruction does not issue at the same time as the instruction immediately before it.
Advanced load address table (ALAT)	The ALAT holds the state necessary for advanced load and check operations.
Translation lookaside buffer (TLB)	The TLB holds virtual to physical address mappings
Virtual hash page table (VHPT)	The VHPT is an extension of the TLB hierarchy, which resides in the virtual memory space, is designed to enhance virtual address translation performance.
Hardware page walker (HPW)	The HPW is the third level of address translation. It is an engine that performs page look-ups from the VHPT and seeks opportunities to insert translations into the processor TLBs.
Register stack engine (RSE)	The RSE moves registers between the register stack and the backing store in memory.
Event address registers (EARs)	The EARs record the instruction and data addresses of data cache misses.

1.2 Related Documentation

The reader of this document should also be familiar with the material and concepts presented in the following documents:

- *Intel® Itanium® Architecture Software Developer's Manual, Volume 1: Application Architecture*
- *Intel® Itanium® Architecture Software Developer's Manual, Volume 2: System Architecture*
- *Intel® Itanium® Architecture Software Developer's Manual, Volume 3: Instruction Set Reference*

§

2 The Dual-Core Itanium 2 Processor

2.1 Overview

The first dual-core Itanium 2 processor, code named Montecito, is the fourth generation of the Itanium 2 processor. Montecito builds on the strength of the previous Itanium 2 processors while bringing many new key technologies for performance and management to the Itanium processor family. Key improvements include multiple-cores, multiple-threads, cache hierarchy, and speculation with the addition of new instructions.

This document describes key Montecito features and how Montecito differs in its implementation of the Itanium architecture from previous Itanium 2 processors. Some of this information may not be directly applicable to performance tuning, but is certainly needed to better understand and interpret changes in application behavior on Montecito versus other Itanium architecture-based processors. Unless otherwise stated, all of the restrictions, rules, sizes, and capacities described in this document apply specifically to Montecito and may not apply to other Itanium architecture-based processors. This document assumes a familiarity with the previous Itanium 2 processors and some of the unique properties and behaviors of those. Furthermore, only differences as they relate to performance will be included here. Information about Montecito features such as error protection, Virtualization technology, Hyper-Threading technology, and lockstep support may be obtained in separate documents.

General understanding of processor components and explicit familiarity with Itanium processor instructions are assumed. This document is not intended to be used as an architectural reference for the Itanium architecture. For more information on the Itanium architecture, consult the *Intel® Itanium® Architecture Software Developer's Manual*.

2.1.1 Identifying the Dual-Core Itanium 2 Processor

There have now been four generations of the Itanium 2 processor, which can be identified by their unique CPUID values. For simplicity of documentation, throughout this document we will group all processors of like model together. Table 2-1details out the CPUID values of all of the Itanium processor family generations. Table 2-2 lists out all of the varieties of the Itanium processor family that are available along with their grouping.

Note that the Montecito CPUID family value changes to 0x20.

Table 2-1. Itanium® Processor Family and Model Values

Family	Model	Description
0x07	0x00	Itanium® Processor
0x1f	0x00	Itanium 2 Processor (up to 3 MB L3 cache)
0x1f	0x01	Itanium 2 Processor (up to 6 MB L3 cache)
0x1f	0x02	Itanium 2 Processor (up to 9 MB L3 cache)
0x20	0x00	Dual-Core Itanium 2 Processor (Montecito)

Table 2-2. Definition Table

Processor	Abbreviation
Intel® Itanium® 2 Processor 900 MHz with 1.5 MB L3 Cache	Itanium 2 Processor (up to 3 MB L3 cache)
Intel® Itanium® 2 Processor 1.0 GHz with 3 MB L3 Cache	
Low Voltage Intel® Itanium® 2 Processor 1.0 GHz with 1.5 MB L3 Cache	Itanium 2 Processor (up to 6 MB L3 cache)
Intel® Itanium® 2 Processor 1.40 GHz with 1.5 MB L3 Cache	
Intel® Itanium® 2 Processor 1.40 GHz with 3 MB L3 Cache	
Intel® Itanium® 2 Processor 1.60 GHz with 3 MB L3 Cache	
Intel® Itanium® 2 Processor 1.30 GHz with 3 MB L3 Cache	
Intel® Itanium® 2 Processor 1.40 GHz with 4 MB L3 Cache	
Intel® Itanium® 2 Processor 1.50 GHz with 6 MB L3 Cache	
Low Voltage Intel® Itanium® 2 Processor 1.30 GHz with 3 MB L3 Cache	Itanium 2 Processor (up to 9 MB L3 cache)
Intel® Itanium® 2 Processor 1.60 GHz with 3 MB L3 Cache at 400 and 533 MHz System Bus (DP Optimized)	
Intel® Itanium® 2 Processor 1.50 GHz with 4 MB L3 Cache	
Intel® Itanium® 2 Processor 1.60 GHz with 6 MB L3 Cache	
Intel® Itanium® 2 Processor 1.60 GHz with 9 MB L3 Cache	
Intel® Itanium® 2 Processor 1.66 GHz with 6 MB L3 Cache	
Intel® Itanium® 2 Processor 1.66 GHz with 9 MB L3 Cache	
Individual SKUs TBD	Dual-Core Itanium 2 Processor (Montecito)

2.1.2 Introducing Montecito

Montecito takes the latest Itanium 2 processor core, improves the memory hierarchy and adds an enhanced form of temporal multi-threading. A full introduction to the Itanium 2 processor is available elsewhere but a brief review is provided below.

The front-end, with two levels of branch prediction, two TLBs, and a 0 cycle branch predictor, feeds two bundles of three instructions each into the instruction buffer every cycle. This 8 entry queue decouples the front-end from the back-end and delivers up to two bundles, of any alignment, to the remaining 6 stages of the pipeline. The dispersal logic determines issue groups and allocates up to 6 instructions to nearly every combination of the 11 available functional units (2 integer, 4 memory, 2 floating point, and 3 branch). The renaming logic maps virtual registers into physical registers. Actual register (up to 12 integer and 4 floating point) reads are performed just before the instructions execute or requests are issued to the cache hierarchy. The full bypass network allows nearly immediate access to previous instruction results while final results are written into the register file (up to 6 integer and 4 floating point).

Montecito preserves application and operating system investments while providing greater opportunity for code generators to continue their steady performance push without any destructive disturbance. This is important since even today, three years after the introduction of the first Itanium 2 processor, compilers are providing significant performance improvements. The block diagram of the Montecito processor can be found in Figure 2-1.

Montecito provides a second integer shifter and popcounter to help reduce port asymmetries. The front-end provides better branching behavior for single cycle branches and cache allocation/ reclamation. Finally, Montecito decreases the time to reach recovery code when speculation fails

thereby providing a lower cost for speculation. All told, nearly every core block and piece of control logic includes some optimization to improve small deficiencies.

Exposing additional performance in an already capable cache hierarchy is also challenging and includes additional capacity, improved coherence architecture, and more efficient cache organization and queuing. Montecito supports three levels of on-chip cache. The first level (L1) caches are each 4-way set associative caches and hold 16 KB of instruction or data. These caches are in-order, like the rest of the pipeline, but are non-blocking allowing high request concurrency. These L1 caches are accessed in a single cycle using pre-validated tags. The data cache is write-through and dual-ported to support two integer loads and two stores, while the instruction cache has dual-ported tags and a single data port to support simultaneous demand and prefetch accesses.

While previous generations of the Itanium 2 processor share the second level (L2) cache with both data and instructions, Montecito provides a dedicated 1 MB L2 cache for instructions. This cache is 8-way set associative with a 128 byte line size and provides the same 7 cycle instruction access latency as the previous smaller Itanium 2 processor unified cache. A single tag and data port supports out-of-order and pipelined accesses to provide a high utilization. The separate instruction and data L2 caches provide more efficient access to the caches compared to Itanium 2 processors where instruction requests would contend against data accesses for L2 bandwidth against data accesses and potentially impact core execution as well as L2 throughput.

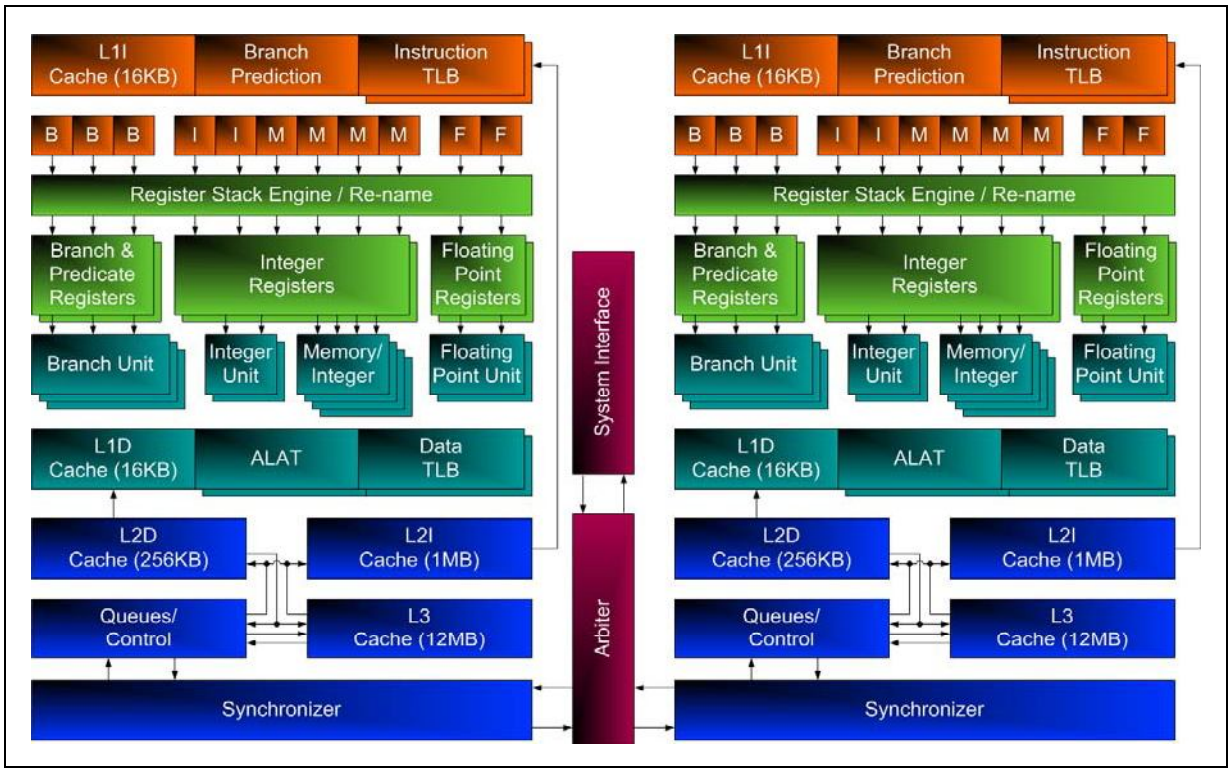
This previously shared 256 KB L2 cache is now dedicated to data on Montecito with several micro-architectural improvements to increase throughput. The instruction and data separation effectively increase the data hit rate. The L2D hit latency remains at 5 cycles for integer and 6 cycles for floating-point accesses. The tag is true 4-ported and the data is pseudo 4-ported with 16-byte banks. Montecito removes some of the code generator challenges found in the Itanium 2 processor L2 cache. Specifically, any accesses beyond the first access to miss the L2 in previous Itanium 2 processors would access the L2 tags periodically until a hit is detected. The repeated tag accesses consume bandwidth from the core and increase the miss latency. On Montecito, such misses are suspended until the L2 fill occurs. The fill awakens and immediately satisfies the request which greatly reduces bandwidth contention and final latency. The Montecito L2D, like previous generations of the Itanium 2 processor L2, is out-of-order and pipelined with the ability to track up to 32 requests in addition to 16 misses and their associated victims. However, Montecito optimizes allocation of the 32 queue entries providing a higher concurrency level than previously possible.

The third level (L3) cache remains unified as in previous Itanium 2 processors, but is now 12 MB in size while maintaining the same 14 cycle integer access latency found on the 6 MB and 9 MB Itanium 2 processors. The L3 uses an asynchronous interface with the data array to achieve this low latency; there is no clock, only a read or write valid indication. The read signal is coincident with index and way values that initiate L3 data array accesses. Four cycles later, the entire 128-byte line is available and latched. This data is then delivered in 4 cycles to either the L2D or L2I cache in critical byte order.

The L3 receives requests from both the L2I and L2D but gives priority to the L2I request in the rare case of a conflict. Moving the arbitration point from the L1-L2 in the Itanium 2 processor to the L2-L3 cache greatly reduces conflicts thanks to the high hit rates of the L2.

The cache hierarchy is replicated in each core to total more than 13.3 MB for each core and nearly 27 MB for the entire processor.

Figure 2-1. The Montecito Processor



2.2 New Instructions

Montecito is compliant with the latest revisions of the Itanium architecture in addition to the *Intel Itanium Architecture Virtualization Specification Update*. As such, Montecito introduces several new instructions as summarized below:

Table 2-3. New Instructions Available in Montecito

New Instruction	Comment
fc.i[1]	Insures that instruction caches are coherent with data caches
ld16[2]	AR.csd and the register specified are the targets for this load.
st16[2]	AR.csd and the value in the register specified are written for this store
cmp8xchg16[2]	AR.csd and the value in the register specified are written for this exchange if the 8 byte compare is true.
hint@pause[3]	The current thread is yielding resources to the other thread.
vmsw.0, vmsw.1	On promote pages, these instructions allow cooperative operating systems to obtain and give up VMM privilege

NOTES:
1. This instruction behaves as the `fc` instruction on Montecito
2. This instruction will fault if issued to UC, UCE, or WC memory
3. This instruction will not initiate a thread switch if it is a B type instruction.

2.3 Core

The Montecito core is very similar to previous generations of the Itanium 2 processor core from a code generation point of view. The core has new resources; specifically, an additional integer shifter and popcounter. The core also removes the rarely needed MMU to Memory Address bypass path. The core also includes many optimizations, from the front-end to the cache hierarchy, that are transparent to the code generator and legacy code can see improvements without any code change.

2.3.1 Instruction Slot to Functional Unit Mapping

This information is very similar to previous Itanium 2 processors. Changes between Itanium 2 processors and Montecito will be noted with footnotes.

Each fetched instruction is assigned to a functional unit through an issue port. The numerous functional units share a smaller number of issue ports. There are 11 functional units: eight for non-branch instructions and three for branch instructions. They are labeled M0, M1, M2, M3, I0, I1, F0, F1, B0, B1, and B2. The process of mapping instructions within bundles to functional units is called *dispersal*.

An instruction's type and position within the issue group determine which functional unit the instruction is assigned. An instruction is mapped to a subset of the functional units based upon the instruction type (i.e. ALU, Memory, Integer, etc.). Then, based on the position of the instruction within the instruction group presented for dispersal, the instruction is mapped to a particular functional unit within that subset.

Table 2-4, Table 2-5, Table 2-6 and Table 2-7 show the mappings of instruction types to ports and functional units.

Note: Shading in the following tables indicates the instruction type can be issued on the port(s).

A-type instructions can be issued on all M and I ports (M0-M3 and I0 and I1). I-type instructions can only issue to I0 or I1. The I ports are asymmetric so some I-type instructions can only issue on port I0. M ports have many asymmetries: some M-type instructions can issue on all ports; some can only issue on M0 and M1; some can only issue on M2 and M3; some can only issue on M0; some can only issue on M2.

Table 2-4. A-Type Instruction Port Mapping

Instruction Type	Description	Examples	Ports
A1-A5	ALU	add, shladd	M0-M3, I0, I1
A4, A5	Add Immediate	addp4, addl	M0-M3, I0, I1
A6,A7,A8	Compare	cmp, cmp4	M0-M3, I0, I1
A9	MM ALU	pcmp[1 | 2 | 4]	M0-M3, I0, I1
A10	MM Shift and Add	pshladd2	M0-M3, I0, I1

Table 2-5. B-Type Instruction Port Mapping

Instruction Type	Description	Examples	Ports
B1-B5	Branch	br	B0-B2
B6-8	Branch Predict	brp	B0-B2
B9[1]	Break, nop, thread switch hint	hint	B0-B2

NOTES:
1. hint.b is treated as a nop.b -- it does not have any impact on multi-thread control in Montecito.

Table 2-6. I-Type Instruction Port Mapping

Instruction Type	Description	Examples	I Port	
			I0	I1
I1	MM Multiply/Shift	pmpy2.[l | r], pmpyshr2{.u}		
I2	MM Mix/Pack	mix[1 | 2 | 4].[l | r pmin, pmax		
I3, I4	MM Mux	mux1, mux2		
I5	Variable Right Shift	shr{.u} =ar,ar pshr[2 | 4] =ar,ar		
I6	MM Right Shift Fixed	pshr[2 | 4] =ar,c		
I7	Variable Left Shift	shl{.u} =ar,ar pshl[2 | 4] =ar,ar		
I8	MM Left Shift Fixed	pshl[2 | 4] =ar,c		
I9[1]	MM Popcount	popcnt		
I10[1]	Shift Right Pair	shrp		
I11-I17[1]	Extr, Dep Test Nat	extr{.u}, dep{.z} tnat		
I18	Hint	hint.i		
I19	Break, Nop	break.i, nop.i		
I20	Integer Speculation Check	chk.s.i		
I21-28	Move to/from BR/PR/IP/AR	mov =[br | pr | ip | ar] mov [br | pr | ip | ar]=		
I29	Sxt/Zxt/Czx	sxt, zxt, czx		

NOTES:
1. The I1 issue capability is new to Montecito

Table 2-7. M-Type Instruction Port Mapping (Sheet 1 of 2)

Instruction Type	Description	Examples	Memory Port			
			M0	M1	M2	M3
M1, 2, 3	Integer Load	ldsz, ld8.fill				
M4, 5	Integer Store	stsz, st8.spill				

Table 2-7. M-Type Instruction Port Mapping (Sheet 2 of 2)

Instruction Type	Description	Examples	Memory Port			
			M0	M1	M2	M3
M6, 7, 8	Floating-point Load	ldf*fsz*, ldf*fsz*.s, ldf.fill	■	■	■	■
	Floating-point Advanced Load	ldf*fsz*.a, ldf*fsz*.c.[clr | nc]	■	■		
M9, 10	Floating-point Store	stf*fsz*, stf.spill			■	■
M11, 12	Floating-point Load Pair	ldfp*fsz*	■	■		
M13, 14, 15	Line Prefetch	lfetch	■	■		
M16	Compare and Exchange	cmpxchg*sz*.[acq | rel]			■	■
M17	Fetch and Add	fetchadd*sz*.[acq | rel]			■	■
M18	Set Floating-point Reg	setf.[s | d | exp | sig}			■	■
M19	Get Floating-point Reg	getf.[s | d | exp | sig}			■	
M20, 21	Speculation Check	chk.s{.m}			■	■
M22, 23	Advanced Load Check	chk.a[clr | nc]	■	■		
M24	Invalidate ALAT	invala	■	■		
	Mem Fence, Sync, Serialize	fwb, mf{.a}, srlz.[d | i], sync.li	■			
M25	RSE Control	flushrs, loadrs	■			
M26, 27	Invalidate ALAT	invala.e	■	■		
M28	Flush Cache, Purge TC Entry	fc, ptc.e			■	
M29, 30, 31	Move to/from App Reg	mov{.m} *ar*= mov{.m} =*ar*			■	
M32, 33	Move to/from Control Reg	mov *cr*=, mov =*cr*			■	
M34	Allocate Register Stack Frame	alloc			■	
M35, 36	Move to/from Proc. Status Reg	mov psr.[l | um] mov =psr.[l | m]			■	
M37	Break, Nop.m	break.m, nop.m	■	■	■	■
M38, 39, 40	Probe Access	probe.[r | w].{fault}			■	
M41	Insert Translation Cache	itc.[d | i]			■	
M42, 43	Move Indirect Reg Insert TR	mov *ireg*=, move =*ireg*, itr.[d | i]			■	
M44	Set/Reset User/System Mask	sum, rum, ssm, rsm			■	
M45	Purge Translation Cache/Reg	ptc.[d | i | g | ga]			■	
M46	Virtual Address Translation	tak, thash, tpa, ttag			■	
M47	Purge Translation Cache	ptc.e			■	
M48	Thread switch hint	hint	■	■	■	■

2.3.2 Instruction Latencies and Bypasses

Table 2-8 lists the Montecito processor operation latencies.

Table 2-8. Execution with Bypass Latency Summary

Consumer (across) Producer (down)	Qual. Pred.	Branch Pred.	ALU	Load Store Addr	Multi-media	Store Data	Fmac	Fmisc	getf	setf
Adder: add, cmp, cmp4, shrp, extr, dep, tbit, addp4, shladd, shladdp4, zxt, sxt, czx, sum, logical ops, 64-bit immed. moves, movl, post-inc ops (includes post-inc stores, loads, lfetches)	n/a	n/a	1	1	3	1	n/a	n/a	n/a	1
Multimedia	n/a	n/a	3	4 or 8[1]	2	3	n/a	n/a	n/a	3
thash, ttag, tak, tpa, probe[2]			5	6	6	5				
getf[2]	n/a	n/a	5	6	6	5	n/a	n/a	n/a	5
setf[2]	n/a	n/a	n/a	n/a	n/a	6	6	6	6	n/a
Fmac: fma, fms, fnma, fpma, fpms, fpnma, fadd, fnmpy, fsub, fpmpy, fpnmpy, fmpy, fnorm, xma, frcpa, fprcpa, frsqrta, fpsqrta, fcvt, fpcvt	n/a	n/a	n/a	n/a	n/a	4	4	4	4	n/a
Fmisc: fselect, fcmp, fclass, fmin, fmax, famin, famax, fpmin, fpmax, fpamin, fpcmp, fmerge, fmix, fsxt, fpack, fswap, fand, fandcm, for, fxor, fpmerge, fneg, fnegabs, fpabs, fpneg, fpnegabs	n/a	n/a	n/a	n/a	n/a	4	4	4	4	n/a
Integer side predicate write: cmp, tbit, tnat	1	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
FP side predicate write: fcmp	2	1	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
FP side predicate write: frcpa, fprcpa, frsqrta, fpsqrta	2	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Integer Load[3]	n/a	n/a	N	N+1	N+2	N	N	N	N	N
FP Load[4]	n/a	n/a	M+1	M+2	M+3	M+1	M+1	M+1	M+1	M+1
IEU2: move_from_br, alloc	n/a	n/a	2	2	3	2	n/a	n/a	n/a	2
Move to/from CR or AR[5]	n/a	n/a	C	C	C	C	n/a	n/a	n/a	C
Move to pr	1	0	2	2	3	2	n/a	n/a	n/a	n/a
Move indirect[6]	n/a	n/a	D	D	D	D	n/a	n/a	n/a	D

NOTES:
1. The MMU to memory address bypass in Montecito does not exist. If code does not account for the missing bypass, the processor will detect the case and cause a pipeflush to ensure proper separation between the producer and the consumer.
2. Since these operations are performed by the L2D, they interact with the L2D pipeline. These are the minimum latencies but they could be much larger because of this interaction.
3. N depends upon which level of cache is hit: N=1 for L1D, N=5 for L2D, N=14-15 for L3, N=~180-225 for main memory. These are minimum latencies and are likely to be larger for higher levels of cache.
4. M depends upon which level of cache is hit: M=5 for L2D, M=14-15 for L3, M=~180-225 for main memory. These are minimum latencies and are likely to be larger for higher levels of cache. The +1 in all table entries denotes one cycle needed for format conversion.
5. Best case values of C range from 2 to 35 cycles depending upon the registers accessed. EC and LC accesses are 2 cycles, FPSR and CR accesses are 10-12 cycles.
6. Best case values of D range from 6 to 35 cycles depending upon the indirect registers accessed. LREGS, PKR, and RR are on the faster side being 6 cycle accesses.

2.3.3 Caches and Cache Management Changes

Montecito, like the previous Itanium 2 processors, supports three levels of on-chip cache. Each core contains a complete cache hierarchy, with nearly 13.3 Mbytes per core, for a total of nearly 27 Mbytes of processor cache.

Table 2-9. Montecito Cache Hierarchy Summary

Cache	Data Types Supported	Write Through/ Write Back	Data Array Size	Line Size	Ways	Index	Queuing	Minimum /Typical Latency
L1D	Integer	WT	16 KB	64 Bytes	4	VA[11:6]	8 Fills	1/1
L1I	Instruction	NA	16 KB	64 Bytes	4	VA[11:6]	1 Demand + 7 Prefetch Fills	1/1
L2D	Integer, Floating Point	WB	256 KB	128 Bytes	8	PA[14:7]	32 OzQ/ 16 Fills	5/11
L2I	Instruction	NA	1 MByte	128 Bytes	8	PA[16:7]	8	7/10
L3	Integer, Floating Point, Instruction	WB	12 MByte	128 Bytes	12	PA[19:7]	8	14/21

2.3.3.1 L1 Caches

The L1I and L1D caches are essentially unchanged from previous generations of the Itanium 2 processor.

2.3.3.2 L2 Caches

Level 2 caches are both different and similar to the Itanium 2 processor L2 cache. The previous Itanium 2 processor L2 shares both data and instructions, while the Montecito has dedicated instruction (L2I) and data (L2D) caches. This separation of instruction and data caches makes it possible to have dedicated access paths to the caches and thus eliminates contention and eases capacity pressures on the L2 caches.

The L2I cache holds 1 Mbyte; is eight-way set associative; and has a 128-byte line size-yet has the same seven-cycle instruction-access latency as the smaller previous Itanium 2 processor unified cache. The tag and data arrays are single ported, but the control logic supports out-of-order and pipelined accesses. This large cache greatly reduces the number of instruction accesses seen at the L3 cache. Any coherence request to identify whether a cache line is in the processor will invalidate that line from the L2I cache.

The L2D cache has the same structure and organization as the Itanium 2 processor shared 256 KB L2 cache but with several microarchitectural improvements to increase throughput. The L2D hit latency remains at five cycles for integer and six cycles for floating-point accesses. The tag array is true four-ported-four fully independent accesses in the same cycle-and the data array is pseudo four-ported with 16-byte banks.

Montecito optimizes several aspects of the L2D. In the Itanium 2 processor, any accesses to the same cache line beyond the first access that misses L2 will access the L2 tags periodically until the tags detect a hit. The repeated tag accesses consume bandwidth from the core and increase the L2 miss latency. Montecito suspends such secondary misses until the L2D fill occurs. At that point, the fill immediately satisfies the suspended request. This approach greatly reduces bandwidth contention and final latency. The L2D, like the Itanium 2 processor L2, is out of order, pipelined, and tracks 32 requests (L2D hits or L2D misses not yet passed to the L3 cache) in addition to 16 misses and their associated victims. The difference is that Montecito allocates the 32 queue entries more efficiently, which provides a higher concurrency level than with the Itanium 2 processor.

Specifically, the queue allocation policy now supports recovery of empty entries. This allows for greater availability of the L2 OzQ in light of accesses completed out of order.

The L2D also considers the thread identifier when performing ordering such that an ordered request from one thread is not needlessly ordered against another thread's accesses.

2.3.3.3 L3 Cache

Montecito's L3 cache remains unified as in previous Itanium 2 processors, but is now 12 MB. Even so, it maintains the same 14-cycle integer-access best case latency in the 6M and 9M Itanium 2 processors. Montecito's L3 cache uses an asynchronous interface with the data array to achieve this low latency; there is no clock, only a read or write valid indication. Four cycles after a read signal, index, and way, the entire 128-byte line is available and latched. The array then delivers this data in four cycles to either the L2D or L2I in critical-byte order.

Montecito's L3 receives requests from both the L2I and L2D but gives priority to the L2I request in the rare case of a conflict. Conflicts are rare because Montecito moves the arbitration point from the Itanium 2 processor L1-L2 to L2-L3. This greatly reduces conflicts because of L2I and L2D's high hit rates. The I and D arbitration point also reduces conflict and access pressure within the core; L1I misses go directly to the L2I and not through the core. L2I misses contend against L2D request for L3 access.

2.3.3.4 Request Tracking

All L2I and L2D requests are allocated to one of 16 request buffers. Requests are sent to the to the L3 cache and system from these buffers by the tracking logic. A modified L2D victim or partial write may be allocated to one of 8 write buffers. This is an increase of 2 over the Itanium 2 processor. The lifetime of the L2D victim buffers is also significantly decreased to further reduce pressure on them. Lastly, the L3 dirty victim resources has grown by 2 entries to 8 in Montecito.

In terms of write coalescing buffers (WCB), Montecito has 4 128B line WCBs in each core. These are fully shared between threads.

2.4 Threading

The multiple thread concept starts with the idea that the processor has some resources that cannot be effectively utilized by a single thread. Therefore, sharing under-utilized resources between multiple threads will increase utilization and performance. The Montecito processor Hyper-Threading Technology implementation duplicates and shares resources to create two logical processors. All architectural state and some micro-architectural state is duplicated.

The duplicated architectural state (general, floating point, predicate, branch, application, translation, performance monitoring, bank, and interrupt registers) allows each thread to appear as a complete processor to the operating system thus minimizing the changes needed at the OS level. The duplicated micro-architectural state of the return stack buffer and the advanced load address table (ALAT) prevent cross-thread pollution that would occur if these resources were shared between the two logical processors.

The two logical processors share the parallel execution resources (core) and the memory hierarchy (caches and TLBs). There are many approaches to sharing resources that vary from fixed time intervals, temporal multi-threading or TMT, to sharing resources concurrently, simultaneous multi-threading or SMT. The Montecito Hyper-Threading Technology approach blends both approaches such that the cores share threads using a TMT approach while the memory hierarchy shares resources using a SMT approach. The core TMT approach is further augmented with control

hardware that monitors the dynamic behavior of the threads and allocates core resources to the most appropriate thread - an event experienced by the workload may cause a switch before the thread quantum of TMT would cause a switch. This modification of TMT may be termed switch-on-event multi-threading.

2.4.1 Sharing Core Resources

Many processors implementing multi-threading share resources using the SMT paradigm. In SMT, instructions from different threads compete for and share execution resources such that each functional resource is dynamically allocated to an available thread. This approach allocates resources originally meant for instruction level parallelism (ILP), but under-utilized in the single thread case, to exploit thread level parallelism (TLP). This is common in many out-of-order execution designs where increased utilization of functional units can be attained for little cost.

Processor resources may also be shared temporally rather than symmetrically. In TMT, a thread is given exclusive ownership of resources for a small time period. Complexity may be reduced by expanding the time quantum to at least the pipeline depth and thus ensure that only a single thread owns any execution or pipeline resources at any moment. Using this approach to multi-threading, nearly all structures and control logic can be thread agnostic allowing the natural behaviors of the pipeline, bypass, and stall control logic for execution to be leveraged while orthogonal logic controls and completes a thread switch is added. However, this approach also means that a pipeline flush is required at thread switch points.

In the core, one thread has exclusive access to the execution resources (foreground thread) for a period of time while the other thread is suspended (background thread). Control logic monitors the workload's behavior and dynamically decreases the time quantum for a thread that is not likely to make progress. Thus, if the control logic determines that a thread is not making progress, the pipeline is flushed and the execution resources are given to the background thread. This ensures better overall utilization of the core resources over strict TMT and effectively hides the cost of long latency operations such as memory accesses.

A thread switch on Montecito requires 15 cycles from initiation until the background thread retires an instruction. Given the low latency of the memory hierarchy (1 cycle L1D, 5 cycle L2D, and 14 cycle L3) memory accesses are the only potentially stalling condition that greatly exceeds the thread switch time and thus is the primary switch event.

A thread switch also has other side effects such as invalidating the Prefetch Virtual Address Buffer (PVAB) and canceling any prefetch requests in the prefetch pipeline.

2.4.1.1 The Switch Events

There are several events that can lead to a thread switch event. Given that hiding memory latency is the primary motivation for multi-threading, the most common switch event is based on L3 cache misses and data returns. Other events, such as the time-out and forward progress event, provide fairness, while the hint events provide paths for the software to influence thread switches. These events have an impact on a thread's urgency which indicates a thread's ability to effectively use core resources. Each event is described below:

- L3 Cache Miss - An L3 miss by the foreground thread is likely to cause that thread to stall waiting for the return from the system interface. Hence, L3 misses can trigger thread switches subject to thread urgency comparisons. This event decreases the thread’s urgency. Since there is some latency between when a thread makes a request and when it is determined to be an L3 miss, it is possible to have multiple requests from a thread miss the L3 cache before a thread switch occurs.

- L3 Cache Return - An L3 miss data return for the background thread is likely to resolve data dependences and is an early indication of execution readiness, hence an L3 miss data return can trigger thread switch events subject to thread urgency comparisons. This event increases the thread's urgency
- Time-out - Thread-quantum counters ensure fairness in access to the pipeline execution resources for each thread. If the thread-quantum expiration occurs when the thread was not stalled, its urgency is set to a high value to indicate execution readiness prior to the switch event.
- Switch Hint - The Itanium architecture provides the `hint@pause` instruction which can trigger a thread switch to yield execution to the background thread. This allows software to indicate when the current thread has no need of the core resources.
- Low-power Mode - When the active thread has entered into a quiesced low-power mode, a thread switch is triggered to the background thread so that it may continue execution. Similarly, if both threads are in a quiesced low-power state, and the background thread is awakened, a thread switch is triggered.

The L3 miss and data return event can occur for several types of accesses: data or instruction, prefetch or demand, cacheable or uncacheable, or hardware page walker (HPW). A data demand access includes loads, stores, and semaphores.

The switch events are intended to enable the control logic to decide the appropriate time to switch threads without software intervention. Thus, Montecito Hyper-Threading Technology is mostly transparent to the application and the operating system

2.4.1.2 Software Control of Thread Switching

The `hint@pause` instruction is used by software to initiate a thread switch. The intent is to allow code to indicate that it does not have any useful work to do and that its execution resources should be given to the other thread. Some later event, such as an interrupt, may change the work for the thread and should awaken the thread.

The `hint@pause` instruction forces a switch from the foreground thread to the background thread. This instruction can be predicated to conditionally initiate a thread switch. Since the current issue group retires before the switch is initiated, the following code sequences are equivalent:

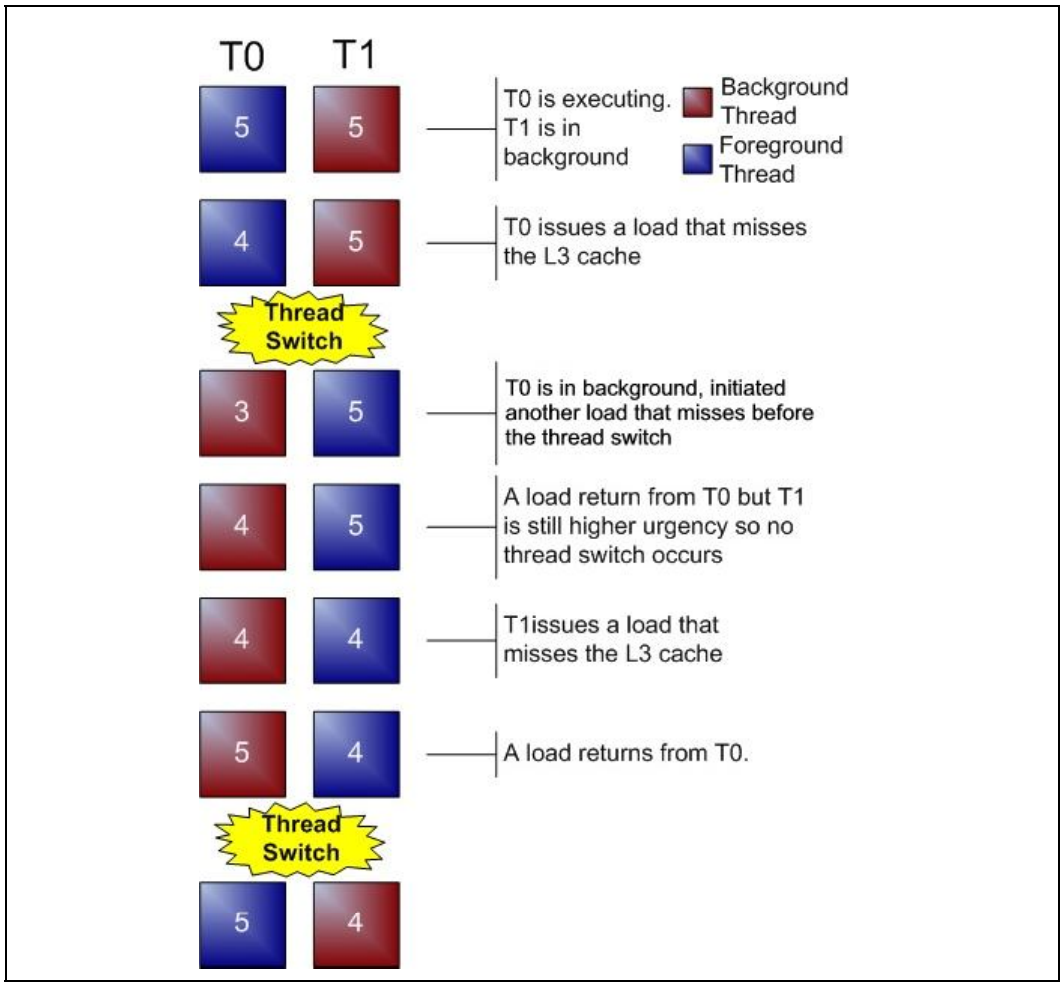
```
{.mii
        hint@pause
        add     r1 = r2, r3
        add     r4 = r2, r0
}

{.mii
        add     r1 = r2, r3
        add     r4 = r2, r0
        hint@pause
}
```

2.4.1.3 Urgency

Each thread has an urgency which can take on values from 0 to 7. A value of 0 denotes that a thread has no useful work to perform. A value of 7 signifies that a thread is actively making forward progress. The nominal urgency is 5 and indicates that a thread is actively progressing. The urgency of one thread is compared against the other at every L3 event. If the urgency of the currently

executing thread is lower than the background thread then the L3 event will initiate a thread switch. Every L3 miss event decrements the urgency by 1, eventually saturating at 0. Similarly, every L3 return event increments the urgency by 1 as long as the urgency is below 5. Figure 2-2 shows a typical urgency based switch scenario. The urgency can be set to 7 for a thread that is switched out due to time-out event. An external interrupt directed at the background thread will set the urgency for that thread to 6 which increases the probability of a thread switch and provide a reasonable response time for interrupt servicing.

Figure 2-2. Urgency and Thread Switching



2.4.2 Tailoring Thread Switch Behavior

Montecito allows the behavior of the thread switch control logic to be tailored to meet specific software requirements. Specifically, thread switch control may emphasize overall performance, thread fairness or elevate the priority of one thread over the other. These different behaviors are available through a low latency PAL call, PAL_SET_HW_POLICY. This will allow software to exert some level of control over how the processor determines the best time to switch. Details on this call and the parameters can be found in the latest *Intel® Itanium® Architecture Software Developer's Manual* and *Intel® Itanium® Architecture Software Developer's Manual Specification Update*.

2.4.3 Sharing Cache and Memory Resources

The Montecito memory resources that are concurrently or simultaneously shared between the two threads include the first and second level TLBs, the first, second, and third level caches, and system interface resources. Each of these structures are impacted in different ways as a result of their sharing.

2.4.3.1 Hyper-Threading Technology and the TLBs

The `ptc.e` instruction in previous Itanium 2 processors would invalidate the entire Translation Cache (TC) section of the TLB with one instruction. This same behavior is retained for Montecito with the caveat that a `ptc.e` issued on one thread will invalidate the TC of the other thread at the same time.

The L2I and L2D TLB on the Itanium 2 processor supported 64 Translation Registers (TR). Montecito supports 32 TRs for each logical processor.

2.4.3.1.1 Instruction TLBs

The replacement algorithms for the L1I and L2I TLB do not consider thread for replacement vector updating. However, the L2I TLB will reserve one TLB entry for each thread to meet the architectural requirements for TCs available to a logical processor.

The TLBs support SMT-based sharing by assigning a thread identifier to the virtual address. Thus, two threads cannot share the same TLB entry at the same time even if the virtual address is the same between the two threads.

Since the L1I TLB is key in providing a pseudo-virtual access to the L1I cache, using prevalidation, when a L1I TLB entry is invalidated, the L1I cache entries associated with that page (up to 4 K) are invalidated. However, the invalidation of a page (and hence cache contents) can be suppressed when two threads access the same virtual and physical addresses. This allows the two threads to share much of the L1I TLB and cache contents. For example, T0 inserts a L1I TLB entry with VA=0 and PA=0x1001000. T0 then accesses VAs 0x000 to 0xFFF which are allocated to the L1I cache. A thread switch occurs. Now, T1 initiates an access with VA=0. It will miss in the L1I TLB because the entry with VA=0 belongs to T0. T1 will insert a L1I TLB entry with VA=0 and PA=0x1001000. The T1 L1I TLB entry replaces the T0 L1I TLB entry without causing an invalidation. Thus, the accesses performed by T0 become available to T1 with the exception of the initial T1 access that inserted the L1I TLB page. Since the L1I cache contents can be shared between two threads and the L1I cache includes branch prediction information, this optimization allows one thread to impact the branch information contained in the L1I cache and hence branch predictions generated for each thread.

2.4.3.1.2 Data TLBs

The replacement algorithms for the L1D and L2D TLB do not consider threads for replacement vector updating. However, the L2D TLB will reserves 16 TLB entries for each thread to meet the architectural requirements for TCs available to a logical processor.

The TLBs support SMT based sharing by assigning a thread identifier to the virtual address. Thus, two threads cannot share the same TLB entry at the same time even if the virtual address is the same between the two threads.

Despite the fact that both the instruction and data L1 TLBs support prevalidation, the L1I TLB optimization regarding cache contents is not supported in the L1D TLB.

2.4.3.2 Hyper-Threading Technology and the Caches

The L2I, L2D, and L3 caches are all physically addressed. Thus, the threads can fully share the cache contents (i.e. an access allocated by T0 can be accessed by both T0 and T1). The queueing resources for these cache levels are equally available to each thread. The replacement logic also ignores threads such that T0 can cause an eviction of T1 allocated data and a hit will cause a cache line to be considered recently used regardless of the thread that allocated or accessed the line.

A thread identifier is provided with each instruction or data cache request to ensure proper ordering of requests between threads at the L2D cache in addition to performance monitoring and switch event calculations at all levels. The thread identifier allows ordered and unordered transactions from T0 pass ordered transactions from T1.

2.4.3.3 Hyper-Threading Technology and the System Interface

The system interface logic also ignores the thread identifier in allocating queue entries and in prioritizing system interface requests. The system interface logic tracks L3 miss and fills and as such, uses the thread identifier to correctly signal to the core which thread missed or filled the cache for L3 miss/return events. The thread identifier is also used in performance monitor event collection and counting.

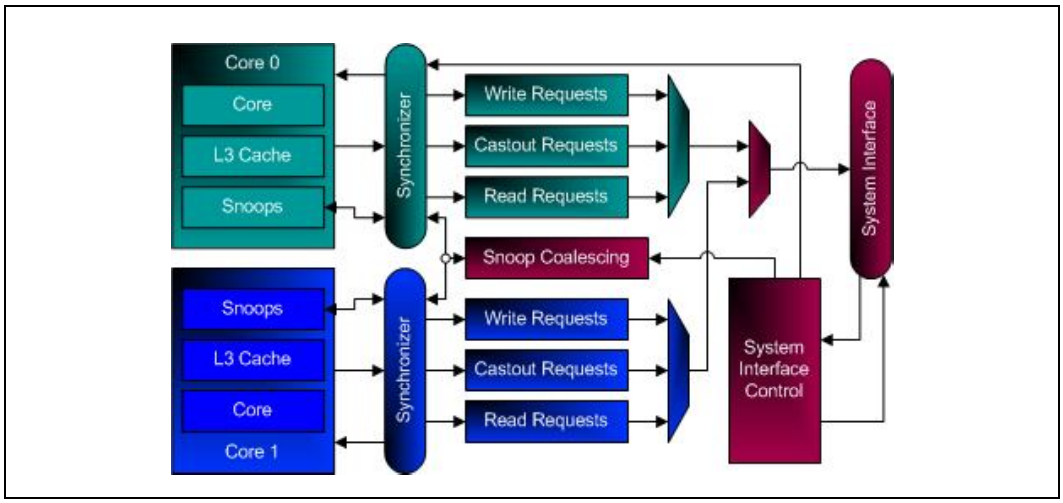
The thread identifier can be made visible on the system interface as part of the agent identifier through a PAL call. This is for informational purposes only as the bit would appear in a reserved portion of the agent identifier and Montecito does not require the memory controller to ensure forward progress and fairness based on the thread identifier -- the L2D cache ensures forward progress between threads.

2.5 Dual Cores

Montecito is the first dual core Itanium 2 processor. The two cores attach to the system interface through the arbiter, which provides a low-latency path for each core to initiate and respond to system events.

Figure 2-3 is a block diagram of the arbiter, which organizes and optimizes each core's request to the system interface, ensures fairness and forward progress, and collects responses from each core to provide a unified response. The arbiter maintains each core's unique identity to the system interface and operates at a fixed ratio to the system interface frequency. The cores are responsible for thread ordering and fairness so the thread identifier to uniquely identify transactions on the system interface is not necessary. However, the processor can be configured to provide the thread identifier for informational purposes only.

Figure 2-3. The Arbiter and Queues



As the figure shows, the arbiter consists of a set of address queues, data queues, and synchronizers, as well as logic for core and system interface arbitration. Error-Correction Code (ECC) encoders/ decoders and parity generators exist but are not shown.

The core initiates one of three types of accesses, which the arbiter allocates to the following queues and buffers:

- Request queue. This is the primary address queue that supports most request types. Each core has four request queues.
- Write address queue. This queue holds addresses only and handles explicit writebacks and partial line writes. Each core has two write address queues.
- Clean castout queue. This queue holds the address for the clean castout (directory and snoop filter update) transactions. The arbiter holds pending transactions until it issues them on the system interface. Each core has four clean castout queues.
- Write data buffer. This buffer holds outbound data and has a one-to-one correspondence with addresses in the write address queue. Each core has four write data buffers, with the additional two buffers holding implicit writeback data.

The number of entries in these buffers are small because they are deallocated once the transaction is issued on the system interface. System interface responses to the transaction are sent directly to the core where the overall tracking of a system interface request occurs.

Note that there are no core to core bypasses present. Thus, a cache line that is requested by core 0 but exists modified on core 1 will be issued to the system interface, snoop core 1 which provides the data and a modified snoop result - all of which is seen on the system interface.

The Snoop queue issues snoop requests to the cores and coalesces the snoop response from each core into a unified snoop response for the socket. If any core is delayed in delivering its snoop response, the arbiter will delay the snoop response on the system interface.

The arbiter delivers all data returns directly to the appropriate core using a unique identifier provided with the initial request. It delivers broadcast transactions, such as interrupts and TLB purges, to both cores in the same way that delivery would occur if each core were connected directly to the system interface.

2.5.1 Fairness and Arbitration

The arbiter interleaves core requests on a one-to-one basis when both cores have transactions to issue. When only one core has requests, its can issue its requests without the other core having to issue a transaction. Because read latency is the greatest concern, the read requests are typically the highest priority, followed by writes, and finally clean castouts. Each core tracks the occupancy of the arbiter's queues using a credit system for flow control. As requests complete, the arbiter informs the appropriate core of the type and number of deallocated queue entries. The cores use this information to determine which, if any, transaction to issue to the arbiter.

2.6 Intel® Virtualization Technology

The Montecito processor is the first Itanium 2 processor to implement Intel® Virtualization Technology. The full specification as well as further information on Intel Virtualization Technology can be found at:
http://www.intel.com/technology/computing/vptech/.

2.7 Tips and Tricks

2.7.1 Cross Modifying Code

Section 2.5 in Part 2 of Volume 2 of the *Intel® Itanium® Architecture Software Developer's Manual* specifies specific sequences that must be followed when any instruction code may exist in the data cache. Many violations of this code may have worked in previous Itanium 2 processors, but such violations are likely to be exposed by the cache hierarchy found in Montecito. Code in violation of the architecture should be modified to adhere to the architectural requirements.

The large L2I and the separation of the instruction and data at the L2 level also requires additional time to ensure coherence if using the PAL_CACHE_FLUSH procedure with the I/D coherence option. Care should be taken to ensure that previously lower cost uses of the PAL_CACHE_FLUSH call should be replaced with the architecture required code sequence for ensuring instruction and data consistency.

2.7.2 ld.bias and lfetch.excl

The `ld.bias` and `lfetch.excl` instructions have been enhanced on the Montecito processor. These instructions can now bring in lines into the cache in a state that is ready to be modified if supported by the memory controller. This feature allows a single `lfetch.excl` or `ld.bias` to prefetch both the source and destination streams. This feature is enabled by default, but may be disabled by PAL_SET_PROC_FEATURES bit 7 of the Montecito feature_set (18).

2.7.3 L2D Victimization Optimization

Montecito also improves on the behaviors associated with internal cache line coherence tracking. The number of false L2D victims will drastically reduce on Montecito over previous Itanium 2 processors. This optimization is enabled by default, but may be disabled by PAL_SET_PROC_FEATURES.

2.7.4 Instruction Cache Coherence Optimization

Coherence requests of the L1I and L2I caches will invalidate the line if it is in the cache. Montecito allows instruction requests on the system interface to be filtered such that they will not initiate coherence requests of the L1I and L2I caches. This will allow instructions to be cached at the L1I and L2I levels across multiple processors in a coherent domain. This optimization is enabled by default, but may be disabled by PAL_SET_PROC_FEATURES bit 5 of the Montecito feature_set (18).

2.8 IA-32 Execution

IA-32 execution on the Montecito processor is enabled with the IA-32 Execution Layer (IA-32 EL) and PAL-based IA-32 execution. IA-32 EL is OS-based and is only available after an OS has booted. PAL-based IA-32 execution is available after PAL_COPY_PAL is called and provides IA-32 execution support before the OS has booted. All OSes running on Montecito have a requirement to have IA-32 EL installed. There is no support for PAL-based IA-32 execution in an OS environment.

IA-32 EL is a software layer that is currently shipping with Itanium architecture-based operating systems and will convert IA-32 instructions into Itanium processor instructions via dynamic translation. Further details on operating system support and functionality of IA-32 EL can be found at http://www.intel.com/cd/ids/developer/asmo-na/eng/strategy/66007.htm.

2.9 Brand Information

One of the newer additions to the Itanium architecture is the PAL_BRAND_INFO procedure. This procedure, along with PAL_PROC_GET_FEATURES, allows software to obtain processor branding and feature information. Details on the above functions can be found in the *Intel® Itanium® Architecture Software Developer's Manual*.

Below is the table of the implementation-specific return values for PAL_BRAND_INFO. Montecito will implement all three, however previous implementations of the Intel Itanium 2 processor are all unable to retrieve the processor frequency, so requests for these fields will return -6, information not available. Also, previous Itanium 2 processors cannot return system bus frequency speed. Implementation-specific values are expected to start at value 16 and continue until an invalid argument (-2) is returned.

Note: The values returned below are the values that the processor was validated at, which is not necessarily the values that the processor is currently running at.

Table 2-10. PAL_BRAND_INFO Implementation-Specific Return Values

Value	Definition
18	The system bus frequency component (in Hz) of the brand identification string will be returned in the brand_info return argument.
17	The cache size component (in bytes) of the brand identification string will be returned in the brand_info return argument.
16	The frequency component (in Hz) of the brand identification string will be returned in the brand_info return argument.

There are other processor features that may not be included in the brand name above. To obtain information on if that technology or feature has been implemented, the PAL_PROC_GET_GEATURES procedure should be used. Montecito features will be in the Montecito processor feature_set (18).

Table 2-11. Montecito Processor Feature Set Return Values

Value	Definition
18	Hyper-Threading Technology (HT) - This processor supports Hyper-Threading Technology
17	Low Voltage (LV) - This processor is a low power SKU
16	Dual-Processor (DP) - This processor is restricted to two processor (DP) systems

§

3 Performance Monitoring

3.1 Introduction to Performance Monitoring

This chapter defines the performance monitoring features of the Montecito processor. The Montecito processor provides 12 48-bit performance counters per thread, 200+ monitorable events, and several advanced monitoring capabilities. This chapter outlines the targeted performance monitor usage models and defines the software interface and programming model.

The Itanium 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 should be able to be monitored and fed back into the code generation process to better encode observed run-time behavior or to expose higher levels of instruction level parallelism. These measurements will be critical for understanding the behavior of compiler optimizations, the use of architectural features such as speculation and predication, or the effectiveness of microarchitectural structures such as the ALAT, the caches, and the TLBs. These measurements will provide the data to drive application tuning and future processor, compiler, and operating system designs.

The remainder of this chapter is divided into the following sections:

- Section 3.2 discusses how performance monitors are used, and presents various Montecito processor performance monitoring programming models.
- Section 3.3 defines the Montecito processor specific performance monitoring features, structures and registers.

Chapter 4 provides an overview of the Montecito processor events that can be monitored.

3.2 Performance Monitor Programming Models

This section introduces the Montecito processor performance monitoring features from a programming model point of view and describes how the different event monitoring mechanisms can be used effectively. The Montecito 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 3.2.1 discusses the Montecito 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 features of the Itanium architecture, such as predication and speculation, benefit from run-time profile information to optimize instruction schedules. The Montecito processor supports instruction level statistical profiling of branch mispredicts and cache misses. Details of the Montecito processor's profiling support are described in Section 3.2.2

3.2.1 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 3.2.1.1 discusses event rate monitoring.
- **Cycle Accounting:** The cycle breakdown 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 3.2.1.4 discusses cycle accounting.

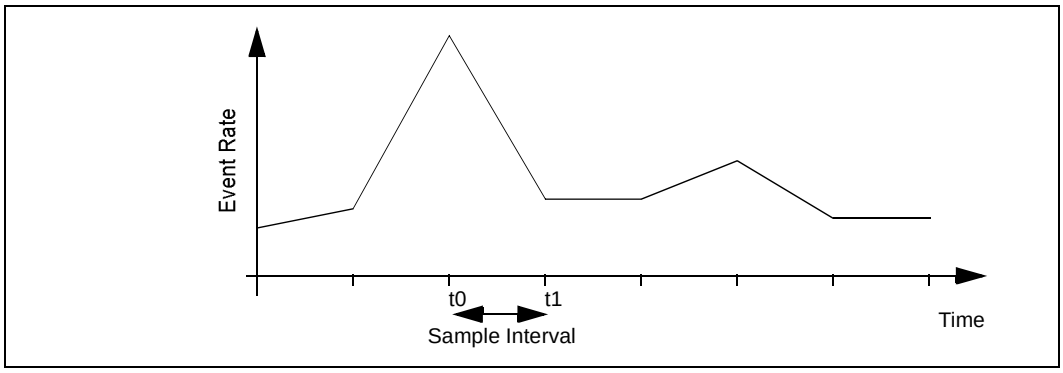
3.2.1.1 Event Rate Monitoring

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 Montecito processor events that count the number of retired Itanium instructions (IA64_INST_RETIRED.u) and the number of elapsed clock cycles (CPU_OP_CYCLES) allow a workload's instructions per cycle (IPC) to be computed as follows:

- IPC = (IA64_INST_RETIRED.u_{t1} - IA64_INST_RETIRED.u_{t0}) / (CPU_OP_CYCLES$_{t1}$ - CPU_OP_CYCLES$_{t0}$)

Time-based sampling is the basis for many performance debugging tools (VTune™ analyzer, gprof, WinNT). As shown in Figure 3-1, time-based sampling can be used to plot the event rates over time, and can provide insights into the different phases that the workload moves through.

Figure 3-1. Time-Based Sampling



On the Montecito 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 Montecito 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 Montecito processor is data cache read 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. Section 3.2.1.2 and Section 3.2.1.3 describe the basic Montecito processor mechanisms for monitoring single and multi-occurrence events.

3.2.1.2 Single Occurrence Events and Duration Counts

A single occurrence event can be monitored by any of the Montecito 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 Montecito processor are TLB misses, branch mispredictions, and cycle-based metrics.

3.2.1.3 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 Montecito processor are retired instructions or the number of live entries in the memory request queue.

Thresholding capabilities are available in the Montecito 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 microarchitectural 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 are 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 the number of live requests per cycle (n_{live}/cycle). By summing up the live requests (n_{live}/cycle) using a multi-occurrence counter, Σn_{live} 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 3-1 in which the average outstanding requests/cycle = 15/8 = 1.825, and the average latency per request = 15/5 = 3 cycles.

Table 3-1. 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 Montecito processor provides the following capabilities to support event rate monitoring:

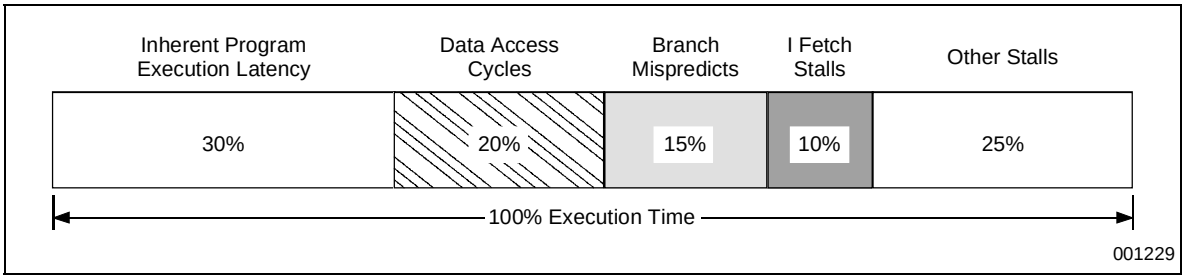
- Clock cycle counter
- Retired instruction counter

- Event occurrence and duration counters
- Multi-occurrence counters with thresholding capability

3.2.1.4 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 Montecito processor provides a set of cycle accounting monitors, that break down the number of cycles that are lost due to various kinds of microarchitectural events. As shown in Figure 3-2, this lets us account for every cycle spent by a program and therefore provides insight into an application's microarchitectural 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 breakdown by stall and flush reasons, while simple duration counters are useful in determining cumulative stall or flush latencies.

Figure 3-2. Itanium® Processor Family Cycle Accounting



The Montecito 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 six back-end stall and flush reasons are prioritized in the following order:

1. Exception/Interruption Cycle: cycles spent flushing the pipe due to interrupts and exceptions.
2. Branch Mispredict Cycle: cycles spent flushing the pipe due to branch mispredicts.
3. Data/FPU Access Cycle: memory pipeline full, data TLB stalls, load-use stalls, and access to floating-point unit.
4. Execution Latency Cycle: scoreboard and other register dependency stalls.
5. RSE Active Cycle: RSE spill/fill stall.
6. Front End Stalls: stalls due to the back-end waiting on the front end.

Additional front-end stall counters are available which detail seven possible reasons for a front-end stall to occur. However, the back-end and front-end stall events should not be compared since they are counted in different stages of the pipeline.

For details, refer to Section 4.6.

3.2.2 Profiling

Profiling is used by application developers, 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 features of the Itanium architecture, 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 Montecito processor's performance monitors:

- Program Counter Sampling
- Miss Event Address Sampling: Montecito processor event address registers (EARs) provide sub-pipeline length event resolution for performance critical events (instruction and data caches, branch mispredicts, and 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 Section 3.2.2.1, Section 3.2.2.2 and Section 3.2.3.3.

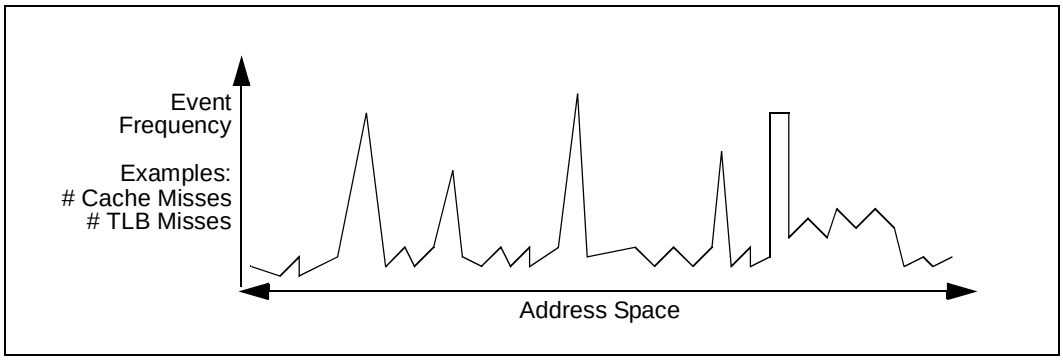
3.2.2.1 Program Counter Sampling

Application tuning tools like VTune analyzer and 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 3-3, 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.

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 Montecito processor, the cycle accounting mechanism (described in Section 3.2.1.4) can be used to directly measure an application's microarchitectural behavior.

The interval timer facilities of the Itanium architecture (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 Section 7.2.2 "Performance Monitor Overflow Status Registers (PMC[0]..PMC[3])" in Volume 2 of the *Intel® Itanium® Architecture Software Developer's Manual*.

Figure 3-3. Event Histogram by Program Counter



To support program counter sampling, the Montecito 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

3.2.2.2 Miss Event Address Sampling

Program counter sampling and cycle accounting provide an accurate picture of cumulative microarchitectural behavior, but they do not provide the application developer with pointers to specific program elements (code locations and data structures) that repeatedly cause microarchitectural “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 microarchitectural “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 superscalar issue, deep pipelining, and out-of-order instruction completion of today’s microarchitectures, 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 two 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 Montecito processor, it is approximately 48 dynamic instructions. If program counter sampling is used for miss event address identification on the Montecito 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 Montecito 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, and the instruction addresses of instruction TLB and cache misses. A 16 entry deep *execution trace buffer* captures sequences of branch instructions and other instructions and events which causes changes to execution flow. Table 3-2 summarizes the capabilities offered by the Montecito processor EARs and the execution 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 3-2. Montecito 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 L1 ITLB (demand fetches only)	Instruction Address What serviced L1 ITLB miss: L2 ITLB 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 What serviced L1 DTLB miss: L2 DTLB, VHPT or software
Execution Trace Buffer	Branch Outcomes rfi, exceptions, failed “chk” instructions which cause a change in execution flow	Source instruction address of the event Target Instruction Address of the event Mispredict status and reason for branches

The Montecito 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 predetermined 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.

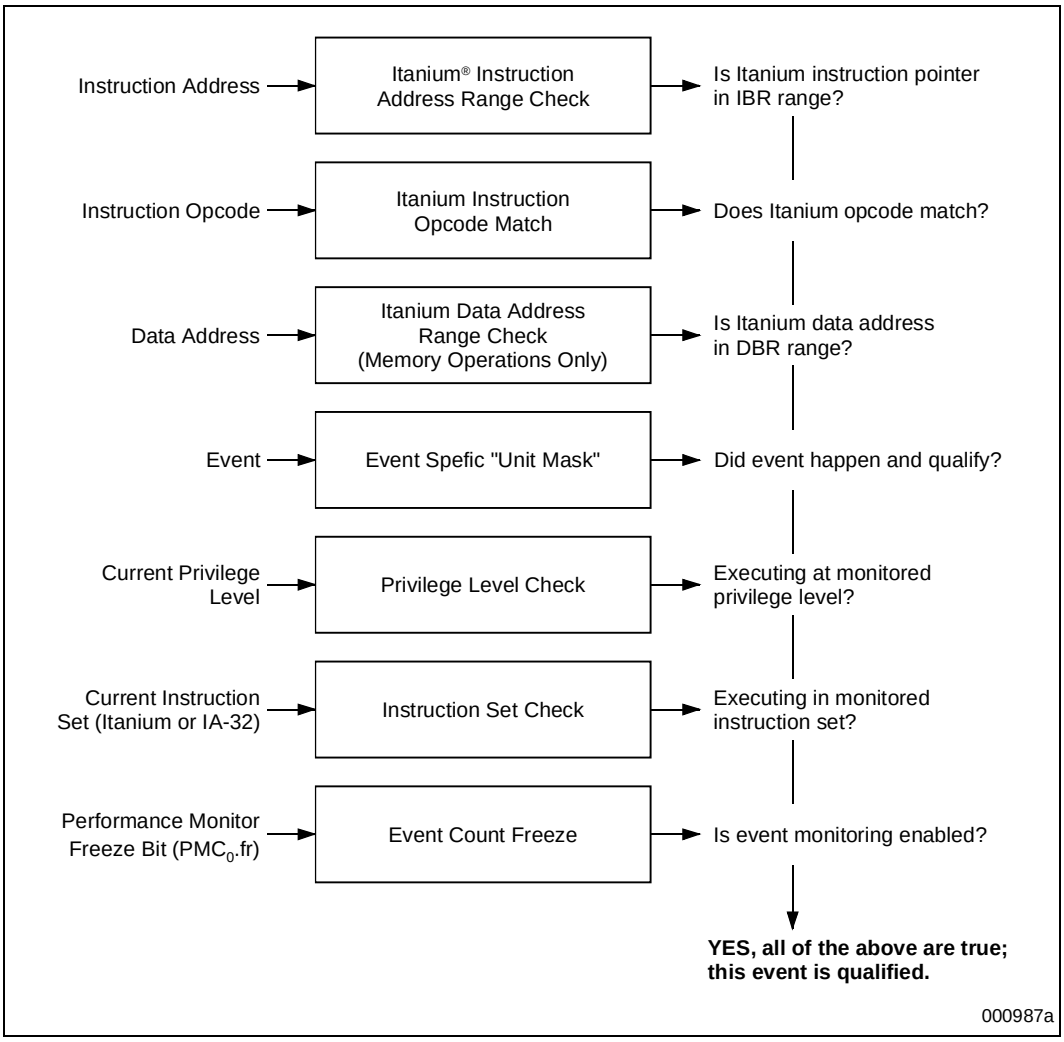
3.2.3 Event Qualification

In the Montecito processor, many of the performance monitoring events can be qualified in a number of ways such that only a subset of the events are counted using performance monitoring counters. As shown in Figure 3-4 events can be qualified for monitoring based on instruction address range, instruction opcode, data address range, event-specific “unit mask” (umask), the privilege level and instruction set the event was caused by, and the status of the performance monitoring freeze bit (PMC_0.fr). The following paragraphs describes these capabilities in detail.

- Itanium Instruction Address Range Check: The Montecito processor allows event monitoring to be constrained to a programmable instruction address range. This enables monitoring of dynamically linked libraries (DLLs), functions, or loops of interest in the context of a large Itanium architecture-based application. The Itanium 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 Montecito processor’s instruction address range check operates only during

Itanium architecture-based code execution, i.e. when PSR.is is zero. For details, see Section 3.3.5.

Figure 3-4. Montecito Processor Event Qualification



- Itanium Instruction Opcode Match: The Montecito processor provides two independent Itanium instruction opcode match ranges, 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 Itanium architecture-based code execution, i.e. when PSR.is is zero. Details are described in Section 3.3.6.
- Itanium Data Address Range Check: The Montecito 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 3.3.7.
- Event Specific Unit Masks: Some events allow the specification of “unit masks” to filter out interesting events directly at the monitored unit. As an example, the number of counted bus transactions can be qualified by an event specific unit mask to contain transactions that

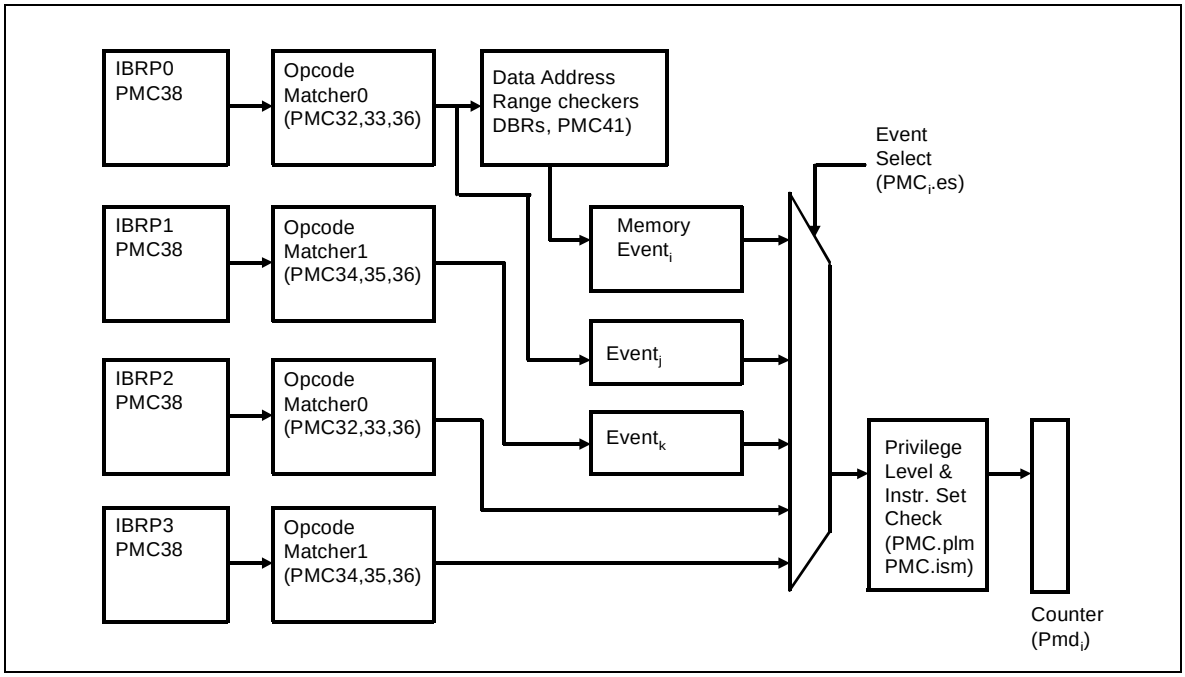
originated from any bus agent, from the processor itself, or from other I/O bus masters. In this case, the bus unit uses a three-way unit mask (any, self, or I/O) that specifies which transactions are to be counted. In the Montecito processor, events from the branch, memory and bus units support a variety of unit masks. For details, refer to the event pages in Chapter 4

- Privilege Level: Two bits in the processor status register(PSR) are provided to enable selective process-based event monitoring. The Montecito 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 3.3.1
- Instruction Set: The Montecito processor supports conditional event counting based on the currently executing instruction set (Itanium 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 Itanium architecture and IA-32 contributions. For details, refer to Section 3.3.1.
- 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. For details refer to Section 7.2.4 of Volume 2 of the *Intel® Itanium® Architecture Software Developer's Manual*.

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

The Montecito processor allows various event qualification mechanisms to be combined by providing the instruction tagging mechanism shown in Figure 3-5.

Figure 3-5. Instruction Tagging Mechanism in the Montecito Processor



During Itanium instruction execution, the instruction address range check is applied first. This is applied separately for each IBR pair (IBRP) to generate 4 independent tag bits which flow down the machine in four tag channels. Tags in the four tag channels are then passed to two opcode matchers that combine the instruction address range check with the opcode match and generate another set of four tags. This is done by combining tag channels 0 and 2 with first opcode match registers and tag channels 1 and 3 with the second opcode match registers as shown in Figure 3-5. Each of the 4 combined tags in the four tag channels can be counted as a retired instruction count event (for details refer to event description "IA64_TAGGED_INST_RETIRED").

Combined Itanium processor address range and opcode match tags in tag channel 0, qualifies all downstream pipeline events. Events in the memory hierarchy (L1 and L2 data cache and data TLB events can further be qualified using a data address DBR RangeTag).

As summarized in Figure 3-5, data address range checking can be combined with opcode matching and instruction range checking on the Montecito processor. Additional event qualifications based on the current privilege level can be applied to all events and are discussed in Section 3.2.3.2.

Table 3-3. Montecito Processor Event Qualification Modes

Event Qualification Modes	Instruction Address Range Check Enable (in Opcode Match) $PMC_{32}.ig\_ad$ (1)	Instruction Address Range Check Config PMC_{38}	Tag Channel Opcode Match Enable PMC_{36}	Opcode Match $PMC_{32,33}$ $PMC_{34,35}$ (2)	Data Address Range Check $[PMC_{41}.e\_dbrp_j$ $PMC_{41}.cfg\_dtag_j]$ (mem pipe events only)
Unconstrained Monitoring, channel 0 (all events)	1	x	x	X	[1,11] or [0,xx]
Unconstrained Monitoring, $channel_i$ (i=0,1,2,3; Limited events only)	0	ig_ibrp_i=1	$Ch_i\_ig\_OPC$=1	X	[1,11] or [0,xx]
Instruction Address Range Check only; channel 0	0	ig_ibrp_0=0	$Ch_0\_ig\_OPC$=1	x	[1,00]
Opcode Matching only $Channel_i$	0	ig_ibrpi=1	$Ch_i\_ig\_OPC$=0	Desired Opcodes	[1,01]
Data Address Range Check only	1	x	x	x	[1,10]
Instruction Address Range Check and Opcode Matching, channel0	0	ig_ibrp_0=0	$Ch_0\_ig\_OPC$=0	Desired Opcodes	[1,01]
Instruction and Data Address Range Check	0	ig_ibrp_0=0	$Ch_0\_ig\_OPC$=1	x	[1,00]
Opcode Matching and Data Address Range Check	0	x	$Ch_0\_ig\_OPC$=0	Desired Opcodes	[1,00]

1. For all cases where PMC32.ig_ad is set to 0, PMC32.inv must be set to 0 if address range inversion is not needed.
2. See column 2 for the value of PMC32.ig_ad bit field.

3.2.3.2 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 impossible. To provide hardware support for this kind of analysis, the Itanium architecture 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 Montecito processor performance monitor control is discussed in Section 3.3.1.

PMC registers can be configured as user-level monitors (PMC_i.pm is 0) or system-level monitors (PMC_i.pm is 1). 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 0, events during interruptions are not counted; if DCR.pp is 1, they are included in the kernel counts.

As shown in Figure 3-6, Figure 3-7, and Figure 3-8, 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 breakdown of event counts can be generated as outlined in the chapter on performance monitoring in the *Intel® Itanium® Architecture Software Developer's Manual*.

3.2.3.3 Instruction Set Constraints

Instruction set constraints are not fully supported in Montecito and the corresponding PMC register instruction set mask (PMCi.ism) should be set to Itanium architecture only ('10) to ensure correct operation. Any other values for these bits may cause undefined behavior.

Figure 3-6. Single Process Monitor

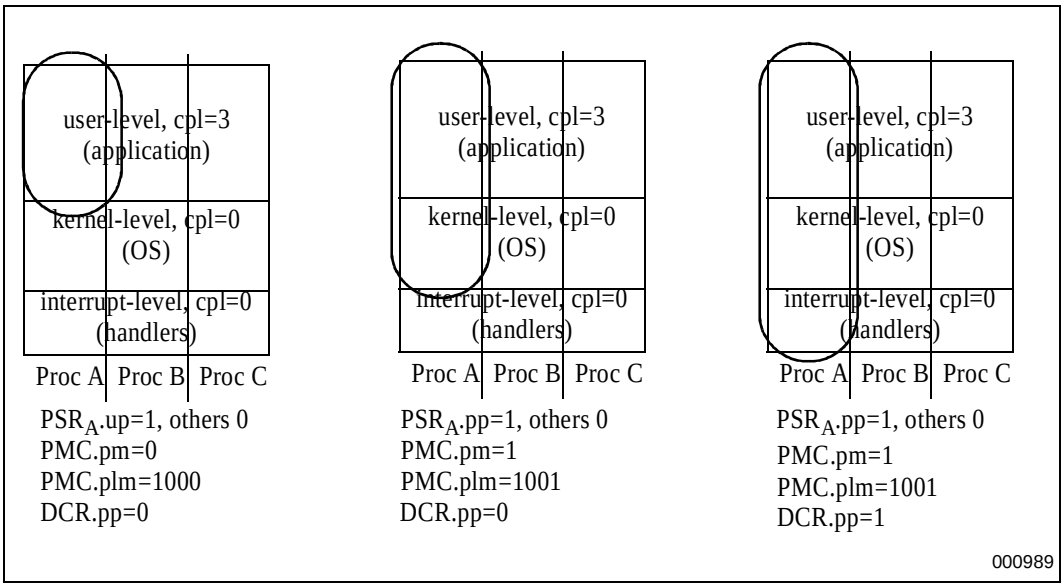


Figure 3-7. Multiple Process Monitor

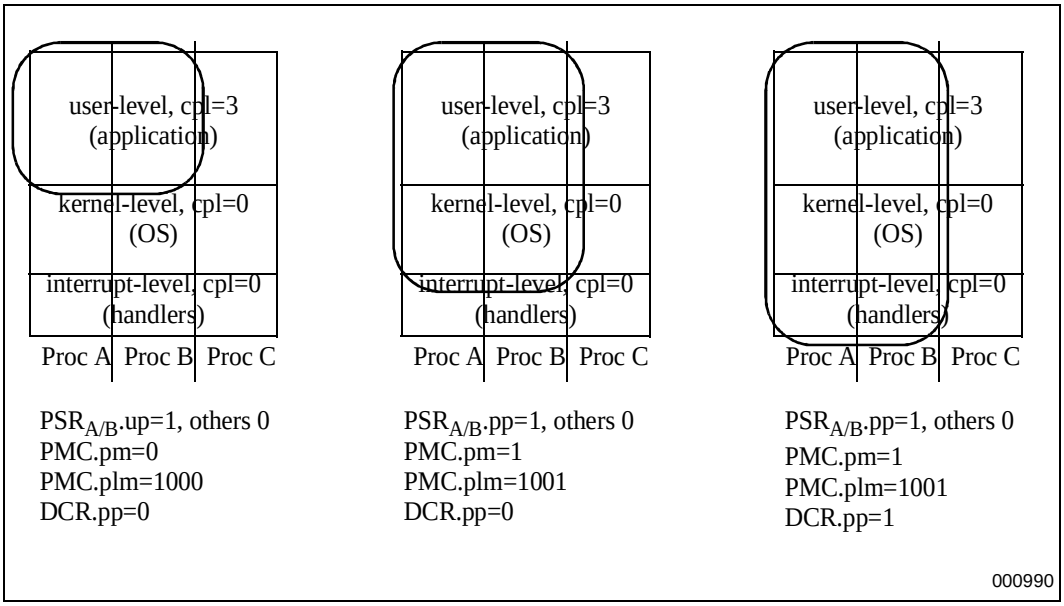
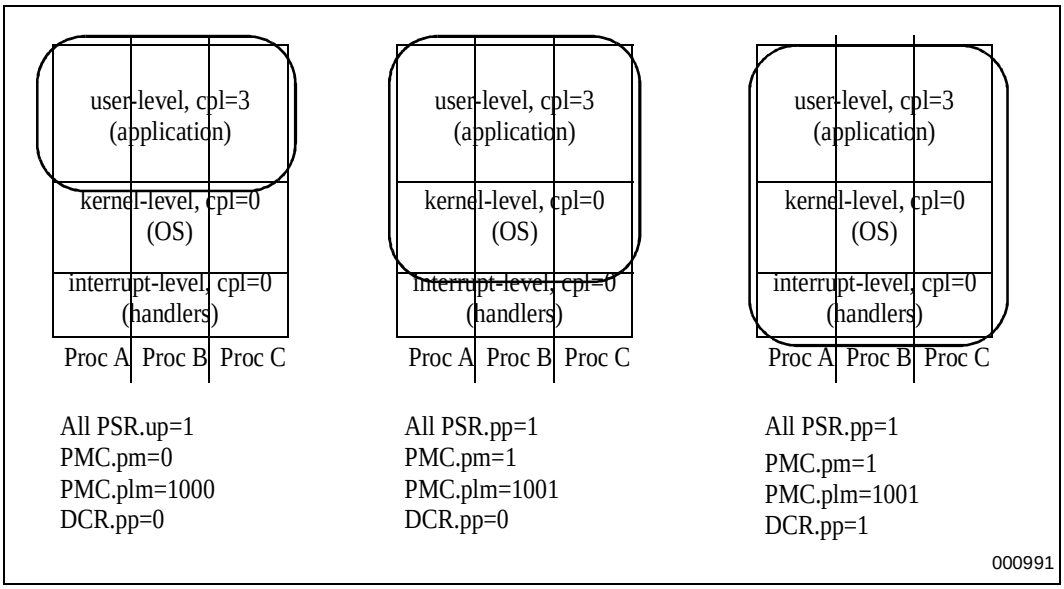


Figure 3-8. System Wide Monitor



3.2.4 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(tm)”, Volume 4 of the Microsoft “Windows NT Resource Kit for Windows NT Version 3.51”, Microsoft Press, 1995.

3.3 Performance Monitor State

Itanium Performance Monitoring architecture described in Volume 2 of the *Intel® Itanium® Architecture Software Developer’s Manual* defines two sets of performance monitor registers; Performance Monitor Configuration (PMC) registers to configure the monitoring and Performance Monitor Data (PMD) registers to provide data values from the monitors. Additionally, the architecture also allows for architectural as well as model specific registers. Complying with this architectural definition, Montecito provides both kind of PMCs and PMDs. As shown in Figure 3-9 the Montecito processor provides 12 48-bit performance counters ($PMC/PMD_{4\text{-}15}$ pairs), and a set of model-specific monitoring registers.

Table 3-4 defines the PMC/PMD register assignments for each monitoring feature. The interrupt status registers are mapped to $PMC_{0,1,2,3}$. The 12 generic performance counter pairs are assigned to $PMC/PMD_{4\text{-}15}$. The Event Address Registers (EARs) and the Execution Trace Buffer (ETB) are controlled by three configuration registers ($PMC_{37,40,39}$). Captured event addresses and cache miss latencies are accessible to software through five event address data registers ($PMD_{34,35,32,33,36}$)

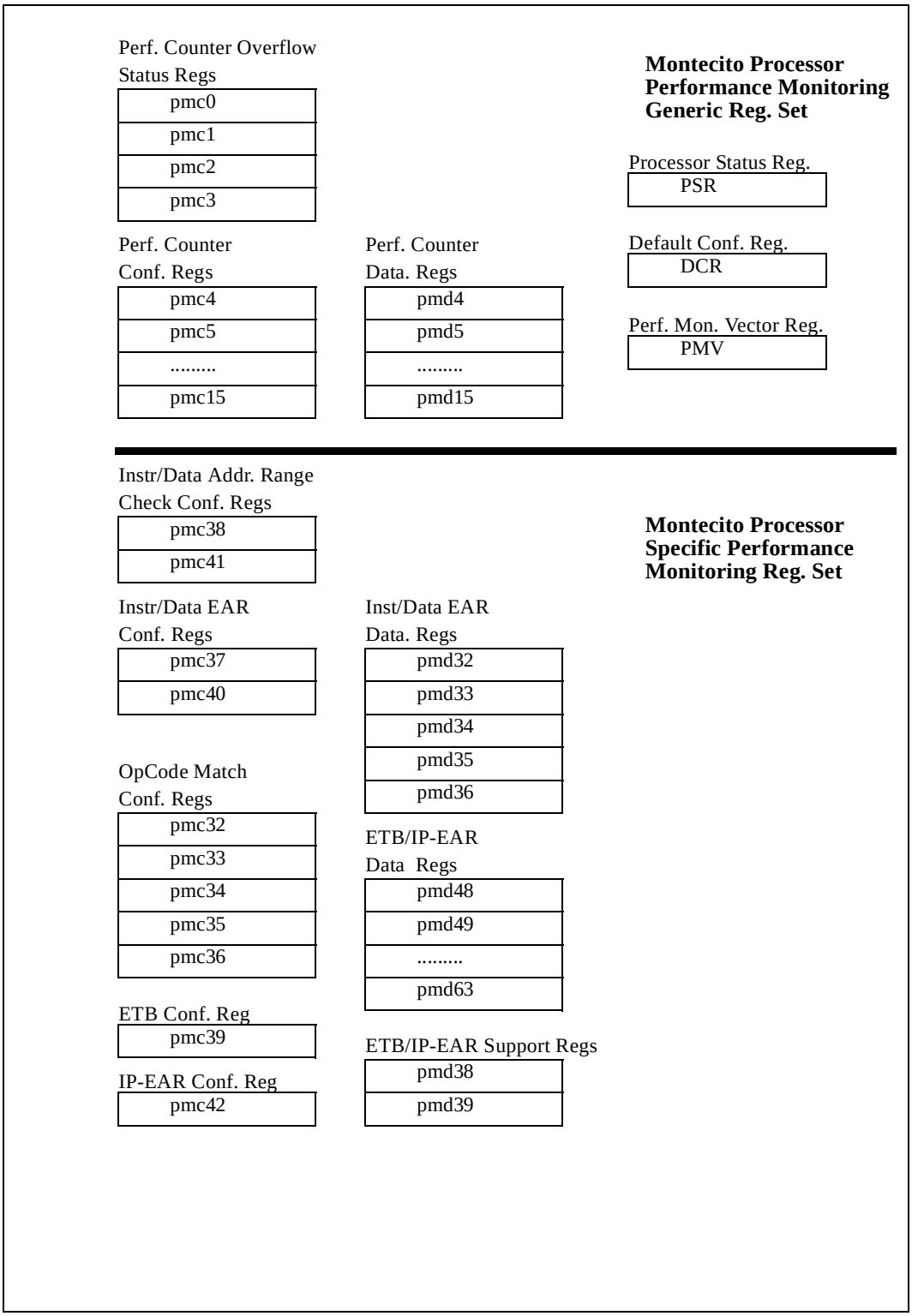
and a branch trace buffer (PMD_{48-63}). On the Montecito processor, monitoring of some events can additionally be constrained to a programmable instruction address range by appropriately setting the instruction breakpoint registers (IBR) and the instruction address range check register (PMC_{38}) and turning on the checking mechanism in the opcode match registers ($PMC_{32,33,34,35}$). Two opcode match register sets and an opcode match configuration register (PMC_{36}) 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 (DBRs) and the data address range configuration register (PMC_{41}).

Montecito, being a processor capable of running two threads, provides the illusion of having two processors by providing exactly the same set of performance monitoring features and structures separately for each thread.

Table 3-4. Montecito Processor Performance Monitor Register Set

Monitoring Feature	Configuration Registers (PMC)	Data Registers (PMD)	Description
Interrupt Status	$PMC_{0,1,2,3}$	none	See Section 3.3.3, “Performance Monitor Event Counting Restrictions Overview”
Event Counters	PMC_{4-15}	PMD_{4-15}	See Section 3.3.2, “Performance Counter Registers”
Opcode Matching	$PMC_{32,33,34,35,36}$	none	See Section 3.3.6, “Opcode Match Check (PMC32,33,34,35,36)”
Instruction EAR	PMC_{37}	$PMD_{34,35}$	See Section 3.3.8, “Instruction EAR (PMC37/ PMD32,33,36)”
Data EAR	PMC_{40}	$PMD_{32,33,36}$	See Section 3.3.9, “Data EAR (PMC40, PMD32,33,36)”
Branch Trace Buffer	PMC_{39}	$PMD_{48-63,39}$	See Section 3.3.10, “Execution Trace Buffer ($PMC_{39,42}$,$PMD_{48-63,38,39}$)”
Instruction Address Range Check	PMC_{38}	none	See Section 3.3.5, “Instruction Address Range Matching”
Memory Pipeline Event Constraints	PMC_{41}	none	See Section 3.3.7, “Data Address Range Matching (PMC41)”
Retired IP EAR	PMC_{42}	$PMD_{48-63,39}$	See Section 3.3.10.2, “IP Event Address Capture (PMC42.mode=‘1xx)”

Figure 3-9. Montecito Processor Performance Monitor Register Mode



3.3.1 Performance Monitor Control and Accessibility

As in other IPF processors, Montecito 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.

Per-monitor control is provided by three PMC register fields (PMC_i.plm, PMC_i.ism, and PMC_i.pm). Event collection for a monitor is enabled under the following constraints on the Montecito processor:

```
Monitor Enable_i =(not PMC_0.fr) and PMC_i.plm[PSR.cpl] and ((PMC_i.ism=í10) or
(i=39)or (i=37)) and ((not(PMC_i.pm) and PSR.up) or (PMC_i.pm and PSR.pp))
```

Figure 3-10 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 and Section 7.2.1 of Volume 2 of the *Intel® Itanium® Architecture Software Developer's Manual*.

Table 3-5 defines per monitor controls that apply to $PMC_{4-15,,32-42}$. As defined in Table 3-4 each of these PMC registers controls the behavior of its associated performance monitor data registers (PMD). The Montecito processor model-specific PMD registers associated with instruction/data EARs and the branch trace buffer ($PMD_{32-39,48-63}$) can be read only when event monitoring is frozen (PMC_0.fr is one).

Figure 3-10. Processor Status Register (PSR) Fields for Performance Monitoring

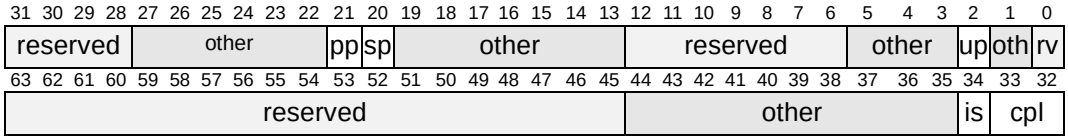


Table 3-5. Performance Monitor PMC Register Control Fields (PMC_{4-15})

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 Montecito 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. Any read of the PMD by non-privileged software in this case will return 0. NOTE: In PMC_{37} this field is implemented in bit [4].
ism	25:24	Instruction Set Mask - Should be set to '10 for proper operation. Undefined behavior with other values. NOTE: PMC_{37} and PMC_{39} do not have this field.

3.3.2 Performance Counter Registers

The PMUs are not shared between hardware threads. Each hardware thread has its own set of 12 generic performance counter (PMC/PMD_{4-15}) pairs.

Due to the complexities of monitoring in an MT "aware" environment, the PMC/PMD pairs are split according to differences in functionality. These PMC/PMD pairs can be divided into two categories; **duplicated** counters (PMC/PMD$_{4-9}$)and **banked** counters (PMC/PMD$_{10-15}$).

- **Banked counters (PMC/PMD$_{10-15}$):** The banked counter capabilities are somewhat limited. These PMDs cannot increment when their thread is in the background. That is, if Thread 0 is placed in the background, Thread 0's PMD$_{10}$ cannot increment until the thread is brought back to the foreground by hardware. Due to this fact, the banked counters should not be used to monitor a thread specific event (.all is set to 0) that could occur when its thread is in the background (e.g. L3_MISSES).
- **Duplicated counters (PMC/PMD$_{4-9}$):** In contrast, duplicated counters can increment when their thread is in the background. As such, they can be used to monitor thread specific events which could occur even when the thread those events belong to is not currently active.

PMC/PMD pairs are not entirely symmetrical in their ability to count events. Please refer to Section 3.3.3 for more information.

Figure 3-11 and Table 3-6 define the layout of the Montecito processor Performance Counter Configuration Registers (PMC$_{4-15}$). The main task of these configuration registers is to select the events to be monitored by the respective performance monitor data counters. Event selection (`es`), unit mask (`umask`), and MESI fields in the PMC registers perform the selection of these events. The rest of the fields in PMCs specify under what conditions the counting should be done (`plm`, `pm`, `ism`), by how much the counter should be incremented (`threshold`), and what need to be done if the counter overflows (`ev`, `oi`).

Figure 3-11. Montecito Processor Generic PMC Registers (PMC$_{4-15}$)

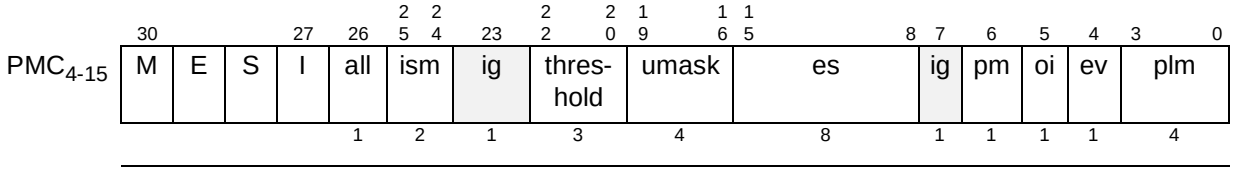


Table 3-6. Montecito Processor Generic PMC Register Fields (PMC$_{4-15}$) (Sheet 1 of 2)

Field	Bits	Description
plm	3:0	Privilege Level Mask. See Table 3-5 "Performance Monitor PMC Register Control Fields (PMC4-15)."
ev	4	External visibility - When 1, an external notification (if the capability is present) is provided whenever the counter overflows. External notification occurs regardless of the setting of the oi bit (see below).
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. Counter overflows generate only one interrupt. Setting the corresponding PMC$_0$ bit on an overflow will be independent of this bit.
pm	6	Privilege Monitor. See Table 3-5 "Performance Monitor PMC Register Control Fields (PMC4-15).".
ig	7	Read zero; writes ignored.
es	15:8	Event select - selects the performance event to be monitored. Montecito processor event encodings are defined in Chapter 4, "Performance Monitor Events."
umask	19:16	Unit Mask - event specific mask bits (see event definition for details)

Table 3-6. Montecito Processor Generic PMC Register Fields (PMC_{4-15}) (Sheet 2 of 2)

Field	Bits	Description
threshold	22:20	Threshold -enables thresholding for “multi-occurrence” events. 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.
ig	23	Read zero, Writes ignored.
ism	25:24	Instruction Set Mask. See Table 3-5 “Performance Monitor PMC Register Control Fields (PMC4-15).”.
all	26	All threads; This bit selects whether or not to monitor just the self thread or both threads. This bit is applicable only for Duplicated counters (PMC4-9) If 1, events from both threads are monitored; If 0, only self thread is monitored. Filters (IAR/DAR/OPC) are only associated with the thread they belong to. If filtering of an event with .all enabled is desired, **both** of the thread’s filters should be given matching configurations.
MESI	30:27	Umask for MESI filtering; Only the events with this capability are affected. [27] : I; [28] = S; [29] = E; [30] = M If the counter is measuring an event implying that a cache line is being replaced, the filter applies to bits in the existing cache line and not the line being brought in. Also note, for the events affected by MESI filtering, if a user wishes to simply captured all occurrences of the event the filter **must** be set to b1111.
ig	63:31	Read zero; writes ignored.

Figure 3-12 and Table 3-7 define the layout of the Montecito processor Performance Counter Data Registers (PMD_{4-15}). A counter overflow occurs when the counter wraps (i.e a carry out from bit 46 is detected). Software can force an external interruption or external notification after N events by preloading the monitor with a count value of 2^{47} - N. Note that bit 47 is the overflow bit and must be initialized to 0 whenever there is a need to initialize the register.

When accessible, software can continuously read the performance counter registers PMD_{4-15} without disabling event collection. Any read of the PMD from software without the appropriate privilege level will return 0 (See “plm” in Table 3-6). The processor ensures that software will see monotonically increasing counter values.

Figure 3-12. Montecito Processor Generic PMD Registers (PMD_{4-15})

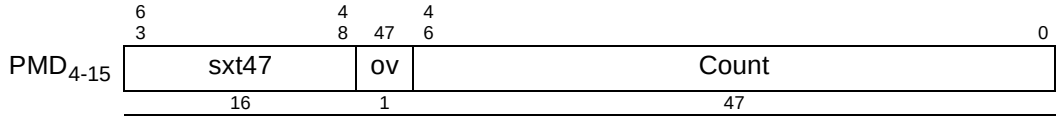


Table 3-7. Montecito Processor Generic PMD Register Fields

Field	Bits	Description
sxt47	63:48	Writes are ignored, Reads return the value of bit 46, so count values appear as sign extended.
ov	47	Overflow bit (carry out from bit 46). NOTE: When writing to a PMD, **always** write 0 to this bit. Reads will return the value of bit 46. DO NOT USE this field to properly determine whether the counter has overflowed or not. Use the appropriate bit from PMC0 instead.
count	46:0	Event Count. The counter is defined to overflow when the count field wraps (carry out from bit 46).

3.3.3 Performance Monitor Event Counting Restrictions Overview

Similar to other Itanium brand products, not all performance monitoring events can be monitored using any generic performance monitor counters (PMD4-15). The following need to be noted when determining which counter to be used to monitor events. This is just an overview and further details can be found under the specific event/event type.

- ER/SI/L2D events can only be monitored using PMD4-9 (These are the events with event select IDs belong to ‘h8x, ‘h9x, ‘hax, ‘hbx, ‘hex and ‘hfx)
- To monitor any L2D events it is necessary to monitor at least one L2D event in either PMC4 or PMC6.(See Section 4.8.4 for more information)
- To monitor any L1D events it is necessary to program PMC5/PMD5 to monitor one L1D event. (See Section 4.8.2 for more information)
- In a MT enabled system, if a “floating” event is monitoring in a banked counter (PMC/PMD_{10-15}), the value may be incorrect. To insure accuracy, these events should be measured by a duplicated counter (PMC/PMD_{4-9}).
- The CYCLES_HALTED event can only be monitored in PMD10. If measured by any other PMD, the count value is undefined.

3.3.4 Performance Monitor Overflow Status Registers ($PMC_{0,1,2,3}$)

As previously mentioned, the Montecito processor supports 12 performance monitoring counters per thread. The overflow status of these 12 counters is indicated in register PMC_0. As shown in Figure 3-13 and Table 3-8 only PMC_0[15:4,0] bits are populated. All other overflow bits are ignored, i.e. they read as zero and ignore writes.

Figure 3-13. Montecito Processor Performance Monitor Overflow Status Registers ($PMC_{0,1,2,3}$)

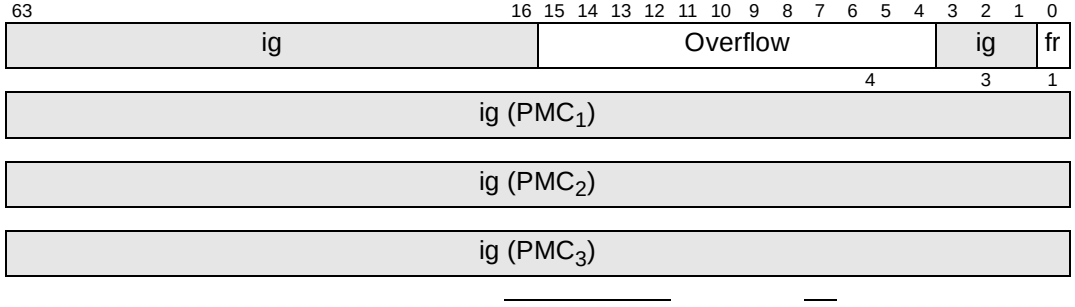


Table 3-8. Montecito Processor Performance Monitor Overflow Register Fields ($PMC_{0,1,2,3}$) (Sheet 1 of 2)

Register	Field	Bits	HW Reset	Description
PMC_0	fr	0	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	ig	3:1	-	Read zero, Writes ignored.

Table 3-8. Montecito Processor Performance Monitor Overflow Register Fields ($PMC_{0,1,2,3}$) (Sheet 2 of 2)

Register	Field	Bits	HW Reset	Description
PMC_0	overflow	15:4	0	Event Counter Overflow - When bit n is one, indicate that the PMDn 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. Software may also set these bits. These bits are sticky and multiple bits may be set.
PMC_0	ig	63:16	-	Read zero, Writes ignored.
$PMC_{1,2,3}$	ig	63:0	-	Read zero, Writes ignored.

3.3.5 Instruction Address Range Matching

The Montecito processor allows event monitoring to be constrained to a range of instruction addresses. Once programmed with this constraints, only the events generated by instructions with their addresses within this range are counted using PMD4-15. The four architectural Instruction Breakpoint Register Pairs $IBRP_{0-3}$ (IBR_{0-7}) are used to specify the desired address ranges. Using these IBR pairs it is possible to define up to 4 different address ranges (only 2 address ranges in "fine mode") that can be used to qualify event monitoring.

Once programmed, each of these 4 address restrictions can be applied separately to all events that are identified to do so. The event, IA64_INST_RETIRED, is the only event that can be constrained using any of the four address ranges. Events described as prefetch events can only be constrained using the address range 2 (IBRP1). All other events can only use the first address range (IBRP0) and this range will be considered as the default for this section.

In addition to constraint events based on instruction addresses, Montecito processor allows event qualification based on the opcode of the instruction and the address of the data the memory related instructions accessed. These are done by applying these constraints to the same 4 instruction address ranges described in this section. These features are explained in Section 3.3.6 and Section 3.3.7.

3.3.5.1 PMC_{38}

Performance Monitoring Configuration register PMC_{38} is the main control register for Instruction Address Range matching feature. In addition to this register, PMC_{32} also controls certain aspects of this feature as explained in the following paragraphs.

Figure 3-14 and Table 3-10 describe the fields of register PMC_{38}. For the proper use of instruction address range checking described in this section, PMC_{38} is expected to be programmed to 0xdb6 as the default value.

Instruction address range checking is controlled by the "ignore address range check" bit (PMC_{32}.ig_ad and PMC_{38}.ig_ibrp0). When PMC_{32}.ig_ad is one (or PMC_{14}.ig_ibrp0 is one), all instructions are included (i.e. un-constrained) regardless of IBR settings. In this mode, events from both IA-32 and Itanium architecture-based code execution contribute to the event count. When both PMC_{32}.ig_ad and PMC_{38}.ig_ibrp0 are zero, the instruction address range check based on the IBRP0 settings is applied to all Itanium processor 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 3-9 defines the behavior of the instruction address range checker for different combinations of `PSR.is` and PMC_{32}.ig_ad or PMC_{38}.ig_ibrp0.

Table 3-9. Montecito Processor Instruction Address Range Check by Instruction Set

	PSR.is	
PMC_{32}.ig_ad OR PMC_{38}.ig_ibrp0	0 (IA-64)	1 (IA-32)
0	Tag only Itanium instructions if they match IBR range	DO NOT tag any IA-32 operations.
1	Tag all Itanium and IA-32 instructions. Ignore IBR range.	

The processor compares every Itanium instruction fetch address IP{63:0} against the address range programmed into the architectural instruction breakpoint register pair $IBRP_0$. Regardless of the value of the instruction breakpoint fault enable (IBR x-bit), the following expression is evaluated for the Montecito processor's $IBRP_0$:

```
IBRmatch = match(IP,IBR0.addr, IBR1.mask, IBR1.plm)
```

The events which occur before the instruction dispersal stage will fire only if this qualified match (IBRmatch) is true. This qualified match will be ANDed with the result of Opcode Matcher $PMC_{32,33}$ and further qualified with more user definable bits (See Table 3-10) before being distributed to different places. The events which occur after instruction dispersal stage, will use this new qualified match (IBRP0-OpCode0 match).

Figure 3-14. Instruction Address Range Configuration Register (PMC_{38})

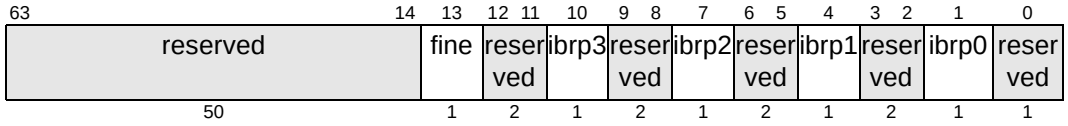


Table 3-10. Instruction Address Range Configuration Register Fields (PMC_{38}) (Sheet 1 of 2)

Field	Bits	Description
ig_ibrp0	1	1: No constraint 0: Address range 0 based on IBRP0 enabled
ig_ibrp1	4	1: No constraint 0: Address range 1 based on IBRP1 enabled
ig_ibrp2	7	1: No constraint 0: Address range2 based on IBRP2 is enabled

Table 3-10. Instruction Address Range Configuration Register Fields (PMC_{38}) (Sheet 2 of 2)

Field	Bits	Description
ig_ibrp3	10	1: No constraint 0: address range 3 based on IBRP3 is enabled
fine	13	Enable fine-mode address range checking (non power of 2) 1: $IBRP_{0,2}$ and $IBRP_{1,3}$ are paired to define two address ranges 0: Normal mode If set to 1, IBRP0 and iIBRP2 define the lower and upper limits for address range0; Similarly, IBRP1 and IBRP3 define the lower and upper limits for address range1. Bits [63:16] of upper and lower limits need to be exactly the same but could have any value. Bits[15:0] of upper limit needs to be greater than bits[15:0] of lower limit. If an address falls in between the upper and lower limits then a match will be signaled only in address ranges 0 or 1. Any event qualification based on address ranges 2 and 3 are not defined. NOTE: The mask bits programmed in IBRs 1,3,5,7 for bits [15:0] have no effect in this mode. When using fine mode address range 0, it is necessary to program PMC38.ig_ibrp0,ig_ibrp2 to 0. Similarly, when using address range 1, it is necessary to set PMC38.ig_ibrp1,ig_ibrp3 to 0.

$IBRP_0$ match is generated in the following fashion. Note that unless fine mode is used, arbitrary range checking cannot be performed since the mask bits are in powers of 2. In fine mode, two IBR pairs are used to specify the upper and lower limits of a range within a page (the upper bits of lower and upper limits must be exactly the same).

```
If PMC38.Fine=0,
    IBRmatch0 = match[IP(63:0), IBR0(63:0), IBR1(55:0)]
Else,
    IBRmatch0 = match[IP(63:16), IBR0(63:16), IBR1(55:16)] and
                 [IP(15:0) > IBR0(15:0)] and [IP(15:0) < IBR4(15:0)]
IBRadrmatch0 = IBRmatch0
ibrp0 match = (PMC32.ig_ad or PMC38.ig_ibrp0) or
               (IBRadmatch0 and match[PSR.cpl, IBR1(59:56)])
```

The instruction range checking considers the address range specified by IBRPi only if PMC_{32}.ig_ad(for i=0), PMC_{38}.ig_ibrp_i and $IBRP_i$ x-bits are all 0s. If the IBRPi x-bits is set, this particular IBRP would be used for debug purposes as described in IA64 architecture.

3.3.5.2 Use of IBRP0 For Instruction Address Range Check - Exception 1

The address range constraint for prefetch events is on the target address of these events rather than the address of the prefetch instruction. Therefore $IBRP_1$ must be used for constraining these events. Calculation of $IBRP_1$ match is the same as that of $IBRP_0$ match with the exception that we use $IBR_{2,3,6}$ instead of $IBR_{0,1,4}$.

Note: Register PMC_{38} must contain the predetermined value 0x0db6. If software modifies any bits not listed in Table 3-10 processor behavior is not defined. It is illegal to have PMC_{41}[48:45]=0000 and PMC_{32}.ig_ad=0 and ((PMC_{38}[2:1]=10 or 00) or (PMC_{38}[5:4]=10 or 00)); this produces inconsistencies in tagging I-side events in L1D and L2.

3.3.5.3 Use of IBRP0 For Instruction Address Range Check - Exception 2

The Address Range Constraint for IA64_TAGGED_INST_RETIRED event uses all four IBR pairs. Calculation of $IBRP_2$ match is the same as that of $IBRP_0$ match with the exception that $IBR_{4,5}$ (in non-fine mode) are used instead of IBR_0. Calculation of $IBRP_3$ match is the same as that of $IBRP_1$ match with the exception that we use $IBR_{6,7}$ (in non-fine mode) instead of $IBR_{2,3}$.

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 Montecito processor pipeline. As described in Section 3.2.3.1, the instruction range check result may be combined with the results of the IA-64 opcode match registers described in Section 3.3.5.4.

3.3.5.4 Fine Mode Address Range Check

In addition to providing coarse address range checking described above, Montecito processor can be programmed to perform address range checks in the fine mode. Montecito provides the use of two address ranges for fine mode. The first range is defined using IBRP0 and IBRP2 while the second is defined using IBRP1 and IBRP3. When properly programmed to use address range 0, all performance monitoring events that has been indicated to be able to qualify with IBRP0 would now qualify with this new address range (defined collectively by IBRP0 and IBRP2). Similarly, when using the address range 1, all events that could be qualified with IBRP1, now get qualified with this new address range.

A user can configure the Montecito PMU to use fine mode address range 0 by following these steps: (It is assumed that PMCs 32,33,34,35,36,38,41 all start with default settings):

- Program IBRP0 and IBRP2 to define the instruction address range. Note to follow the programming restrictions mentioned in Table 3-10
- Program PMC32[ig_ad,inv] = ‘00 to turn off default tags injected into tag channel 0
- Program PMC38[ig_ibrp0,ig_ibrp2] = ‘00 to turn on address tagging based on IBRP0 and IBRP2.
- Program PMC38.fine = 1

Similarly, a user can configure Montecito PMU to use fine mode address range by following the same steps as above but this time with IBRP1&3. The only exception is that PMC32[ig_ad,inv] need not to be programmed.

3.3.6 Opcode Match Check ($PMC_{32,33,34,35,36}$)

As shown in Figure 3-5, in the Montecito processor, event monitoring can be constrained based on the Itanium processor encoding (opcode) of an instruction. Registers $PMC_{32,33,34,35,36}$ allow configuring this feature.In Montecito, registers $PMC_{32,33}$ and $PMC_{34,35}$ define 2 opcode matchers (Opcode matcher 0 (OpCM0) and Opcode Matcher 1 (OpCM1)). Register PMC_{36} controls how to apply opcode range checking to the four instruction address ranges defined by using IBRPs.

3.3.6.1 PMC32,33,34,35

Figure 3-15, Figure 3-16 and Table 3-11, Table 3-12 describe the fields of $PMC_{32,33,34,35}$ registers. Figure 3-17 and Table 3-14 describes the register PMC_{36}.

All combinations of bits [51:48] in $PMC_{32,34}$ are supported. To match a A-slot instruction, it is necessary to set bits [51:50] to 11. To match all instruction types, bits [51:48] should be set to 1111. To ensure that all events are counted independent of the opcode matcher, all mifb and all mask bits of $PMC_{32,34}$ should be set to one (all opcodes match) while keeping the inv bit cleared.

Once the opcode matcher constraints are generated, they are ANDed with the address range constraints available on 4 IBRP channels to form 4 combined address range and opcode match ranges as described here. The constraints defined by OpCM0 are ANDed with address constraints defined by IBRP0 and IBRP2 to form combined constraints for channels 0 and 2. Similarly, the constraints defined by OpCM1 are ANDed with address constraints defined by $IBRP_1$ and $IBRP_3$ to form combined constraints for channels 1 and 3.

Figure 3-15. Opcode Match Registers ($PMC_{32,34}$)

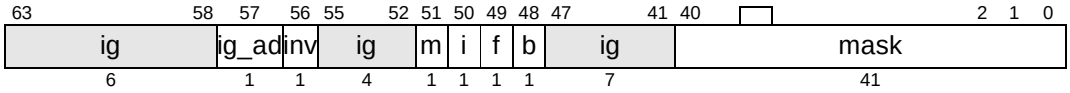


Table 3-11. Opcode Match Registers($PMC_{32,34}$)

Field	Bits	Width	HW Reset	Description
mask	40:0	41	all 1	Bits that mask Itanium® instruction encoding bits. Any of the 41 syllable bits can be selectively masked If mask bit is set to 1, the corresponding opcode bit is not used for opcode matching
ig	47:41	7	n/a	Reads zero; Writes ignored
b	48	1	1	If 1: match if opcode is an B-slot
f	49	1	1	If 1: match if opcode is an F-slot
i	50	1	1	If 1: match if opcode is an I-slot
m	51	1	1	If 1: match if opcode is an M-slot
ig	55:52	4	n/a	Reads zero; writes ignored
inv	56	1	1	Invert Range Check. for tag channel 0 If set to 1, the address ranged specified by IBRP0 is inverted. Effective only when ig_ad bit is set to 0. NOTE: This bit is ignored in PMC_{34}
ig_ad	57	1	1	Ignore Instruction Address Range Checking for tag channle0 If set to 1, all instruction addresses are considered for events. If 0, IBRs 0-1 will be used for address constraints. NOTE: This bit is ignored in PMC_{34}
ig	63:58	4	n/a	Reads zero; Writes ignored

Figure 3-16. Opcode Match Registers ($PMC_{33,35}$)

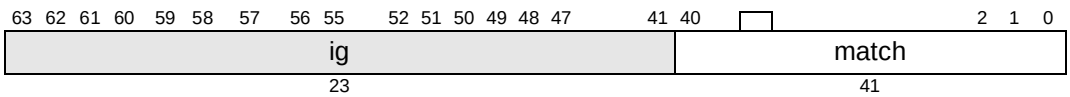


Table 3-12. Opcode Match Registers($PMC_{33,35}$)

Field	Bits	Width	HW Reset	Description
match	40:0	41	all 1s	Opcode bits against which Itanium® instruction encoding to be matched. Each opcode bit has a corresponding bit position here.
ig	63:41	23	n/a	Ignored bits

3.3.6.2 PMC36

Performance Monitoring Configuration register PMC36 controls whether or not to apply opcode matching in event qualification. As mentioned earlier, opcode matching is applied to the same four instruction address ranges defined by using IBRPs.

Figure 3-17. Opcode Match Configuration Register (PMC_{36})

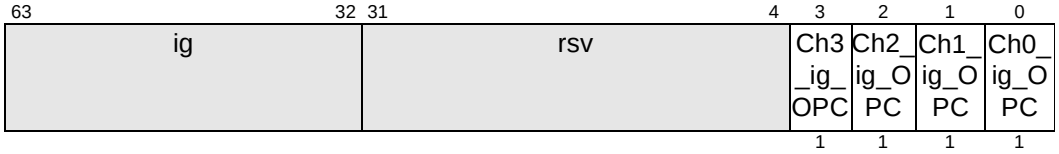


Table 3-13. Opcode Match Configuration Register Fields (PMC_{36})

Field	Bits	HW Reset	Description
Ch0_ig_OPC	0	0	1: Tag channel0 PMU events will not be constrained by opcode 0: Tag channel0 PMU events (including IA64_TAGGED_INST_RETIRED.00) will be opcode constrained by OpCM0
Ch1_ig_OPC	1	0	1: tag channle1 events (IA64_TAGGED_INST_RETIRED.01) won't be constrained by opcode 0: tag channel1 events will be opcode constrained by OpCM1
Ch2_ig_OpC	2	0	1: Tag channel2 events (IA64_TAGGED_INST_RETIRED.10) won't be constrained by opcode 0: Tag channel2 events will be opcode constrained by OpCM0
Ch3_ig_OpC	3	0	1: Tag channel3 events (IA64_TAGGED_INST_RETIRED.11) won't be constrained by opcode 0: Tag channel2 events will be opcode constrained by OpCM1
rsv	31:4	0xffffffff	Reserved. Users should not change this field from reset value
ig	63:32	n/a	Ignored bits

For opcode matching purposes, an Itanium instruction is defined by two items: the instruction type "itype" (one of M, I, F or B) and the 41-bit encoding "enco{40:0}" defined the *Intel® Itanium® Architecture Software Developer's Manual*. Each instruction is evaluated against each opcode match register (OpCM0 and OpCM1) as follows:

```
Match(OpCM[i]) = (imatch(itype, OpCM[i].mifb) AND
ematch(enco,OpCM[i].match,OpCM[i].mask))
```

Where:

```
imatch(itype,OpCMi.mifb) = (itype=M AND PMC[32+i].m) OR (itype=I AND PMC[32+i].i)
OR (itype=F AND PMC[32+i].f) OR (itype=B AND PMC[32+i].b)
ematch(enco,match,mask) = AND b=40..0 ((enco{b}=match{b}) OR mask{b})
```

The IBRP matches are advanced with the instruction pointer to the point where opcodes are being dispersed. The matches from opcode matchers are ANDed with the IBRP matches at this point.

This produces two opcode match events that are combined with the instruction range check tag (IBRRangeTag, see Section 3.3.5) as follows:

```
Tag(IBRChnl0) = Match(OpCM0) and IBRRangeTag0
Tag(IBRChnl1) = Match(OpCM1) and IBRRangeTag1
Tag(IBRChnl2) = Match(OpCM0) and IBRRangeTag2
Tag(IBRChnl3) = Match(OpCM1) and IBRRangeTag3
```

As shown in Figure 3-5 the 4 tags, Tag(IBRChnli; i=0-3) are staged down the processor pipeline until instruction retirement and can be selected as a retired instruction count event (see event description "IA64_TAGGED_INST_RETIRED"). In this way, a performance counter (PMC/PMD_{4-15}) can be used to count the number of retired instructions within the programmed range that match the specified opcodes.

Note: Register PMC_{36} must contain the predetermined value of 0xfffffff0. If software modifies any bits not listed in Table 3-13 processor behavior is not defined. This is the reset value for PMC36.

3.3.7 Data Address Range Matching (PMC_{41})

For instructions that reference memory, the Montecito processor allows event counting to be constrained by data address ranges. The 4 architectural Data Breakpoint Registers (DBRs) can be used to specify the desired address range. Data address range checking capability is controlled by the Memory Pipeline Event Constraints Register (PMC_{41}).

Figure 3-18 and Table 3-14 describe the fields of register $PMC_{41.}$ When enabled ([1,x0] in the bits corresponding to one of the 4 DBRs to be used), data address range checking is applied to loads, stores, semaphore operations, and the `lfetch` instruction.

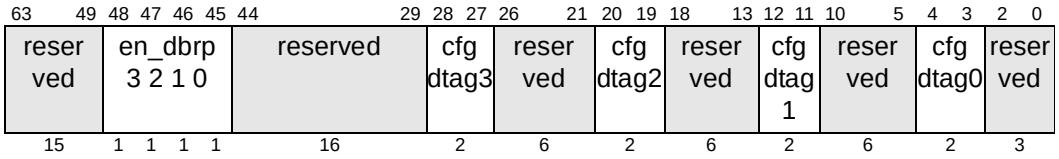
Table 3-14. Memory Pipeline Event Constraints Fields (PMC_{41}) (Sheet 1 of 2)

Field	Bits	Description
cfgdtag0	4:3	These bits determine whether and how $DBRP_0$ should be used for constraining memory pipeline events (where applicable) 00: IBR/Opc/DBR - Use $IBRP_0$/OpCM0 and $DBRP_0$ for constraints (i.e. they will be counted only if their Instruction Address, opcodes and Data Address matches the $IBRP_0$ programmed into these registers) 01: IBR/Opc - Use $IBRP_0$/OpCM0 for constraints 10: DBR - Only use $DBRP_0$ for constraints 11: No constraints NOTE: When used in conjunction with "fine" mode (see PMC14 description), only the lower bound DBR Pair (DBRP0 or DBRP1) config needs to be set. The upper bound DBR Pair config should be left to no constraint. So if IBRP0,2 are chosen for "fine" mode, cfgdtag0 needs to be set according to the desired constraints but cfgdtag2 should be left as 11 (No constraints).
cfgdtag1	12:11	These bits determine whether and how $DBRP_1$ should be used for constraining memory pipeline events (where applicable); bit for bit these match those defined for $DBRP_0$
cfgdtag2	20:19	These bits determine whether and how $DBRP_2$ should be used for constraining memory pipeline events (where applicable); bit for bit these match those defined for $DBRP_0$

Table 3-14. Memory Pipeline Event Constraints Fields (PMC_{41}) (Sheet 2 of 2)

Field	Bits	Description
cfgdtag3	48,28:27	These bits determine whether and how $DBRP_3$ should be used for constraining memory pipeline events (where applicable); bit for bit these match those defined for $DBRP_0$
en_dbrp0	45	0 - No constraints 1 - Constraints as set by cfgdtag0
en_dbrp0	46	0 - No constraints 1 - Constraints as set by cfgdtag1
en_dbrp0	47	0 - No constraints 1 - Constraints as set by cfgdtag2
en_dbrp0	48	0 - No constraints 1 - Constraints as set by cfgdtag3

Figure 3-18. Memory Pipeline Event Constraints Configuration Register (PMC_{41})



$DBRP_x$ match is generated in the following fashion. Arbitrary range checking is not possible since the mask bits are in powers of 2. Although it is possible to enable more than one DBRP at a time for checking, it is not recommended. The resulting four matches are combined as follows to form a single DBR match:

```
DBRRangeMatch = (DBRRangeMatch0 or DBRRangeMatch1 or DBRRangeMatch2 or
DBRRangeMatch3)
```

Events which occur after a memory instruction gets to the EXE stage will fire only if this qualified match ($DBRP_x$ match) is true. The data address is compared to $DBRP_x$; the address match is further qualified by a number of user configurable bits in PMC_{41} before being distributed to different places. DBR matching for performance monitoring ignores the setting of the DBR r,w, and plm fields.

In order to allow simultaneous use of some DBRs for Performance Monitoring and the others for debugging (the architected purpose of these registers), separate mechanisms are provided for enabling DBRs. DBR bits x and the r/w-bit should be cleared to 0 for the DBRP which is going to be used for the PMU. PSR.db has no effect when DBRs are used for this purpose.

Note: Register PMC_{41} must contain the predetermined value 0x2078fefefefe. If software modifies any bits not listed in Table 3-14 processor behavior is not defined. It is illegal to have PMC41[48:45]=0000 and PMC_{32}[57]=0 and ((PMC_{38}[2:1]=10 or 00) or (PMC_{38}[5:4]=10 or 00)); this produces inconsistencies in tagging I-side events in L1D and L3.

3.3.8 Instruction EAR (PMC_{37}/$PMD_{32,33,36}$)

This section defines the register layout for the Montecito processor instruction event address registers (IEAR). The IEAR, configured through PMC_{37}, can be programmed in one of two modes: instruction cache and instruction TLB miss collection. EAR specific unit masks allow software to

specify event collection parameters to hardware. Figure 3-19 and Table 3-15 detail the register layout of PMC_{37}. The instruction address, latency and other captured event parameters are provided in three PMD registers ($PMD_{32,33,36}$). Table 3-20 describes the associated event address data registers $PMD_{34,35}$.

Both the instruction and data cache EARs (see Section 3.3.9) report the latency of captured cache events and allow latency thresholding to qualify event capture. Event address data registers (PMD_{32-36}) contain valid data only when event collection is frozen (PMC_0.fr is one). Reads of PMD_{32-36} while event collection is enabled return undefined values.

Figure 3-19. Instruction Event Address Configuration Register (PMC_{37})

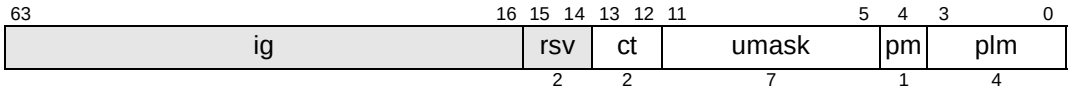
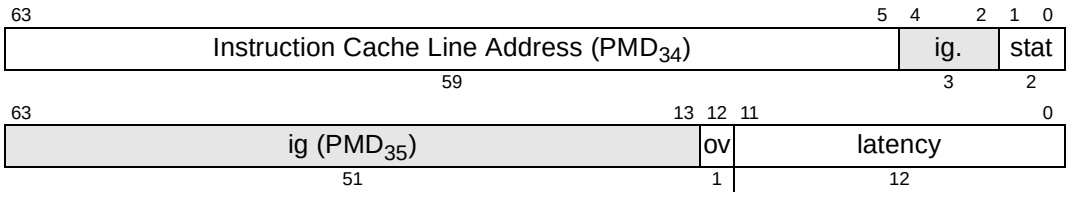


Table 3-15. Instruction Event Address Configuration Register Fields (PMC_{37})

Field	Bits	HW Reset	Description	
plm	3:0	0	See Table 3-5, “Performance Monitor PMC Register Control Fields (PMC4-15)”	
pm	4	0	See Table 3-5, “Performance Monitor PMC Register Control Fields (PMC4-15)”	
umask	11:5 12:5	0	Selects the event to be monitored If [13] = ‘1 then [12:5] are used for umask	
ct	13:12	0	cache_tlb bit. Instruction EAR selector. Select instruction cache or TLB stalls	
			if =1x:	Monitor demand instruction cache misses NOTE: ISB hits are not considered misses. $PMD_{34,35}$ register interpretation (see Table 3-17)
			if =01:	Nothing monitored
			if =00:	Monitor L1 instruction TLB misses $PMD_{34,35}$ register interpretation (see Table 3-17)
rsv	15:14	0	Reserved bits	
ignored	63:16	-	Reads are 0; Writes are ignored	

Figure 3-20. Instruction Event Address Register Format ($PMD_{34,35}$)



When the cache_tlb field (PMC_{37}.ct) is set to 1x, instruction cache misses are monitored. When it is set to 00, instruction TLB misses are monitored. The interpretation of the umask field and performance monitor data registers $PMD_{34,35}$ depends on the setting of this bit and is described in Section 3.3.8.1 for instruction cache monitoring and in Section 3.3.8.2 for instruction TLB monitoring.

3.3.8.1 Instruction EAR Cache Mode (PMC_{37}.ct='1x)

When PMC_{37}.ct is 1x, 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 an eight bit umask field in the configuration register PMC_{37}. Possible threshold values are defined in Table 3-16.

Table 3-16. Instruction EAR (PMC_{37}) umask Field in Cache Mode (PMC_{37}.ct='1x)

umask Bits 12:5	Latency Threshold [CPU cycles]	umask Bits 12:5	Latency Threshold [CPU cycles]
01xxxxxx	>0 (All L1 Misses)	11100000	>=256
11111111	>=4	11000000	>=1024
11111110	>=8	10000000	>=4096
11111100	>=16	other	undefined
11111000	>=32	00000000	RAB hit (All L1 misses which hit in RAB)
11110000	>=128		

As defined in Table 3-17, the address of the instruction cache line missed the L1 instruction cache is provided in PMD_{34}. If no qualified event was captured, it is indicated in PMD_{34}.stat. The latency of the captured instruction cache miss in CPU clock cycles is provided in the latency field of PMD_{35}.

Table 3-17. Instruction EAR ($PMD_{34,35}$) in Cache Mode (PMC_{37}.ct='1x)

Register	Field	Bits	Description
PMD34	stat	1:0	Status x0: EAR did not capture qualified event x1: EAR contains valid event data
	Instruction Cache Line Address	63:5	Address of instruction cache line that caused cache miss
PMD_{35}	latency	11:0	Latency in CPU clocks
	overflow	12	If 1, latency counter has overflowed one or more times before data was returned

3.3.8.2 Instruction EAR TLB Mode (PMC_{37}.ct=00)

When PMC_{37}.ct is '00, 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 3-18 summarizes the instruction TLB umask settings. All combinations of the mask bits are supported.

Table 3-18. Instruction EAR (PMC_{37}) umask Field in TLB Mode (PMC_{37}.ct=00) (Sheet 1 of 2)

ITLB Miss Type	PMC.umask[7:5]	Description
---	000	Disabled; nothing will be counted
L2TLB	xx1	L1 ITLB misses which hit L2 TLB
VHPT	x1x	L1 Instruction TLB misses that hit VHPT

Table 3-18. Instruction EAR (PMC_{37}) umask Field in TLB Mode (PMC_{37}.ct=00) (Sheet 2 of 2)

ITLB Miss Type	PMC.umask[7:5]	Description
FAULT	1xx	Instruction TLB miss produced by an ITLB Miss Fault
ALL	111	Select all L1 ITLB Misses NOTE: All combinations are supported.

As defined in Table 3-19 the address of the instruction cache line fetch that missed the L1 ITLB is provided in PMD_{34}. The stat bit [1] indicates whether the captured TLB miss hit in the VHPT or required servicing by software. PMD_{34}.stat will indicate whether a qualified event was captured. In TLB mode, the latency field of PMD_{35} is undefined.

Table 3-19. Instruction EAR ($PMD_{34,35}$) in TLB Mode (PMC_{37}.ct='00)

Register	Field	Bits	Description
PMD_{34}	stat	1:0	Status Bits 00: EAR did not capture qualified event 01: L1 ITLB miss hit in L2 ITLB 10: L1 ITLB miss hit in VHPT 11: L1 ITLB miss produced an ITLB Miss Fault
	Instruction Cache Line Address	63:5	Address of instruction cache line that caused TLB miss
PMD_{35}	latency	11:2	Undefined in TLB mode

3.3.9 Data EAR (PMC_{40}, $PMD_{32,33,36}$)

The data event address configuration register (PMC_{40}) can be programmed to monitor either L1 data cache load misses, FP loads, L1 data TLB misses, or ALAT misses. Figure 3-21 and Table 3-20 detail the register layout of PMC_{40}. Figure 3-22 describes the associated event address data registers $PMD_{32,33,36}$. The mode bits in configuration register PMC_{40} select data cache, data TLB, or ALAT monitoring. The interpretation of the umask field and registers $PMD_{32,33,36}$ depends on the setting of the mode bits and is described in Section 3.3.9.1 for data cache load miss monitoring, Section 3.3.9.2 for data TLB monitoring, and Section 3.3.9.3 for ALAT monitoring.

Both the instruction (see Section 3.3.8) and data cache EARsreport the latency of captured cache events and allow latency thresholding to qualify event capture. Event address data registers (PMD_{32-36}) contain valid data only when event collection is frozen (PMC_0.fr is one). Reads of PMD_{32-36} while event collection is enabled return undefined values.

Figure 3-21. Data Event Address Configuration Register (PMC_{40})

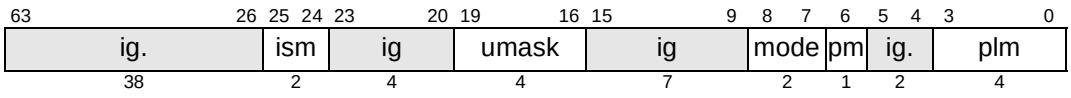


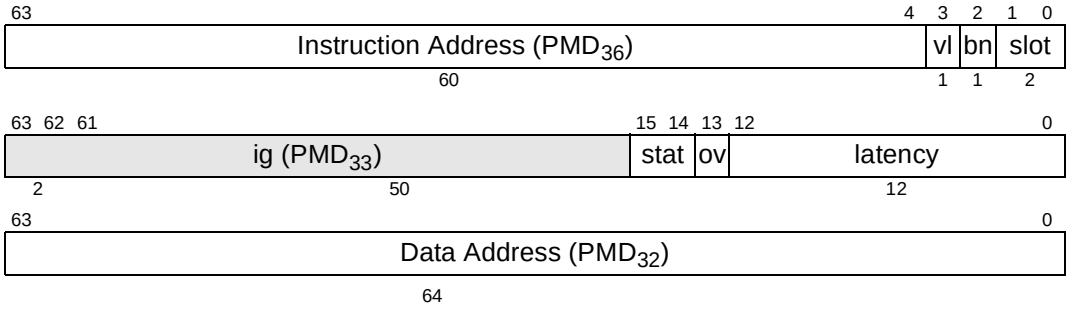
Table 3-20. Data Event Address Configuration Register Fields (PMC_{40}) (Sheet 1 of 2)

Field	Bits	HW Reset	Description
plm	3:0	0	See Table 3-5 "Performance Monitor PMC Register Control Fields (PMC4-15)."
ig	5:4	-	Reads 0; Writes are ignored

Table 3-20. Data Event Address Configuration Register Fields (PMC_{40}) (Sheet 2 of 2)

Field	Bits	HW Reset	Description
pm	6	0	See Table 3-5 “Performance Monitor PMC Register Control Fields (PMC4-15).”
mode	8:7	0	Data EAR mode selector: ‘00: L1 data cache load misses and FP loads ‘01: L1 data TLB misses ‘1x: ALAT misses
ig	15:9	-	Reads 0; Writes are ignored
umask	19:16		Data EAR unit mask mode 00: data cache unit mask (definition see Table 3-21, “Data EAR (PMC40) Umask Fields in Data Cache Mode (PMC40.mode=00)”) mode 01: data TLB unit mask (definition see Table 3-23, “Data EAR (PMC40) Umask Field in TLB Mode (PMC40.ct=01)”)
ig	23:20	-	Reads 0; Writes are ignored
ism	25:24		See Table 3-5 “Performance Monitor PMC Register Control Fields (PMC4-15).”
ig	63:26	-	Reads 0; Writes are ignored

Figure 3-22. Data Event Address Register Format ($PMD_{32,d3,36}$)



3.3.9.1 Data Cache Load Miss Monitoring (PMC_{40}.mode=00)

If the Data EAR is configured to monitor data cache load misses, the umask is used as a load latency threshold defined by Table 3-21.

As defined in Table 3-22, the instruction and data addresses as well as the load latency of a captured data cache load miss are presented to software in three registers $PMD_{2,3,17}$. If no qualified event was captured, the valid bit in PMD_3 is zero.

HPW accesses will not be monitored. `setf` and reads from ccv will not be monitored. If an L1D cache miss is not at least 7 clocks after a captured miss, it will not be captured. Semaphore instructions and floating point loads will be counted.

Table 3-21. Data EAR (PMC_{40}) Umask Fields in Data Cache Mode (PMC_{40}.mode=00)

umask Bits 19:16	Latency Threshold [CPU cycles]	umask Bits 19:16	Latency Threshold [CPU cycles]
0000	>= 4 (Any latency)	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 3-22. $PMD_{32,33,36}$ Fields in Data Cache Load Miss Mode (PMC_{40}.mode=00)

Register	Fields	Bit Range	Description
PMD_{32}	Data Address	63:0	64-bit virtual address of data item that caused miss
PMD_{33}	latency	12:0	Latency in CPU clocks
	overflow	13	Overflow - If 1, latency counter has overflowed one or more times before data was returned
	stat	15:14	Status bits; 00: No valid information in PMD32,36 and rest of PMD33 01: Valid information in PMD32,33 and may be in PMD36 NOTE: These bits should be cleared before the EAR is reused.
	ig	63:26	Reads 0; Writes are ignored
PMD_{36}	slot	1:0	Slot bits; If ".vl" is 1, the Instruction bundle slot of memory instruction
	bn	2	Bundle bit; If ".vl" is 1 this indicates which of the executed bundles is associated with the captured miss
	vl	3	Valid bit; 0: Invalid Address (EAR did not capture qualified event) 1: EAR contains valid event data NOTE: This bit should be cleared before the EAR is reused
	Instruction Address	63:4	Virtual address of the first bundle in the 2-bundle dispersal window which was being executed at the time of the miss. If ".bn" is 1 then the second bundle contains memory instruction and 16 should be added to the address.

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 Montecito 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 $PMD_{32,33,36}$. 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.

3.3.9.2 Data TLB Miss Monitoring (PMC_{40}.mode='01)

If the Data EAR is configured to monitor data TLB misses, the umask defined in Table 3-24 determines which data TLB misses are captured by the Data EAR. For TLB monitoring, all combinations of the mask bits are supported.

As defined in Table 3-24 the instruction and data addresses of captured DTLB misses are presented to software in $PMD_{32,36}$. If no qualified event was captured, the valid bit in PMD_{36} reads zero. When programmed for data TLB monitoring, the contents of the latency field of PMD_{33} are undefined.

Both load and store TLB misses will be captured. Some unreached instructions will also be captured. For example, if a load misses in L1DTLB but hits in L2 DTLB and is in an instruction group after a taken branch, it will be captured. Stores and floating-point operations never miss in L1DTLB but could miss the L2 DTLB or fault to be handled by software.

Note: PMC_{39} must be 0 in this mode; else the wrong IP for misses coming right after a mispredicted branch.

Table 3-23. Data EAR (PMC_{40}) Umask Field in TLB Mode (PMC_{40}.ct=01)

L1 DTLB Miss Type	PMC.umask[19:16]	Description
---	000x	Disabled; nothing will be counted
L2DTLB	xx1x	L1 DTLB misses which hit L2 DTLB
VHPT	x1xx	L1 DTLB misses that hit VHPT
FAULT	1xxx	Data TLB miss produced a fault
ALL	111x	Select all L1 DTLB Misses NOTE: All combinations are supported.

Table 3-24. $PMD_{32,33,36}$ Fields in TLB Miss Mode (PMC_{40}.mode='01) (Sheet 1 of 2)

Register	Field	Bit Range	Description
PMD_{32}	Data Address	63:0	64-bit virtual address of data item that caused miss
PMD_{33}	latency	12:0	Undefined in TLB Miss mode
	ov	13	Undefined in TLB Miss mode
	stat	15:14	Status 00: invalid information in PMD32,36 and rest of PMD33 01: L2 Data TLB hit 10: VHPT hit 11: Data TLB miss produced a fault NOTE: These bits should be cleared before the EAR is reused.
	ig	63:26	Reads 0; Writes are ignored
PMD_{36}	slot	1:0	Slot bits; If ".vl" is 1, the Instruction bundle slot of memory instruction.
	bn	2	Bundle bit; If ".vl" is 1 this indicates which of the executed bundles is associated with the captured miss

Table 3-24. $PMD_{32,33,36}$ Fields in TLB Miss Mode (PMC_{40}.mode='01) (Sheet 2 of 2)

Register	Field	Bit Range	Description
	vl	3	Valid bit; 0: Invalid Instruction Address 1: EAR contains valid instruction address of the miss NOTE: It is possible for this bit to contain 0 while PMD33.stat indicate valid D-EAR data. This can happen when D-EAR is triggered by an RSE load for which no instruction address is captured. NOTE: This bit should be cleared before the EAR is reused.
	Instruction Address	63:4	Virtual address of the first bundle in the 2-bundle dispersal window which was being executed at the time of the miss. If ".bn" is 1 then the second bundle contains memory instruction and 16 should be added to the address.

3.3.9.3 ALAT Miss Monitoring (PMC_{40}.mode='1x)

As defined in Table 3-25, the address of the instruction (failing `chk.a` and `ld.c`) causing an ALAT miss is presented to software in PMD_{36}. If no qualified event was captured, the valid bit in PMD_{36} reads zero. When programmed for ALAT monitoring, the latency field of PMD_{33} and the contents of PMD_{32} are undefined.

Note: PMC_{39} must be 0 in this mode; else the wrong IP for misses coming right after a mispredicted branch.

Table 3-25. $PMD_{32,33,36}$ Fields in ALAT Miss Mode (PMC_{11}.mode='1x)

Register	Field	Bit Range	Description
PMD_{32}	Data Address	63:0	Undefined in ALAT Miss Mode
PMD_{33}	latency	12:0	Undefined in ALAT Miss mode
	ov	13	Undefined in ALAT Miss mode
	stat	15:14	Status bits; 00: No valid information in $PMD_{32,36}$ and rest of PMD_{33} 01: Valid information in $PMD_{32,33}$ and may be in PMD_{36} NOTE: These bits should be cleared before the EAR is reused.
	ig	63:26	Reads 0; Writes are ignored
PMD_{36}	slot	1:0	Slot bits; If ".vl" is 1, the Instruction bundle slot of memory instruction
	bn	2	Bundle bit; If ".vl" is 1 this indicates which of the executed bundles is associated with the captured miss
	vl	3	Valid bit; 0: Invalid Address (EAR did not capture qualified event) 1: EAR contains valid event data NOTE: This bit should be cleared before the EAR is reused.
	Instruction Address	63:4	Virtual address of the first bundle in the 2-bundle dispersal window which was being executed at the time of the miss. If ".bn" is 1 then the second bundle contains memory instruction and 16 should be added to the address.

3.3.10 Execution Trace Buffer ($PMC_{39,42}$,$PMD_{48-63,38,39}$)

The execution trace buffer provides information about the most recent Itanium processor control flow changes. The Montecito execution trace buffer configuration register (PMC_{39}) defines the conditions under which instructions which cause the changes to the execution flow are captured, and allows the trace buffer to capture specific subsets of these events.

In addition to the branches captured in the previous generations of Itanium 2 processor BTB, Montecito's ETB captures rfi instructions, exceptions (excluding asynchronous interrupts) and silently resteered chk (failed chk) events. Passing chk instructions are not captured under any programming conditions (except when there is another capturable event).

In every cycle in which a qualified change to the execution flow happens, its source bundle address and slot number are written to the execution trace buffer. This event's target address is written to the next buffer location. If the target instruction bundle itself contains a qualified execution flow change, the execution trace buffer either records a single trace buffer entry (with the s-bit set) or makes two trace buffer entries: one that records the target instruction as a branch target s-bit cleared), and another that records the target instruction as a branch source (s-bit set). As a result, the branch trace buffer may contain a mixed sequence of the source and target addresses.

Note: The setting of PMC_{42} can override the setting of PMC_{39}. PMC_{42} is used to configure the Execution Trace Buffer's alternate mode: the IP-EAR. Please refer to Section 3.3.10.2.1, "Notes on the IP-EAR" for more information about this mode. PMC_{42}.mode must be set to 000 to enable normal branch trace capture in PMD_{48-63} as described below. If PMC_{42}.mode is set to other than 000, PMC_{39}'s contents will be ignored.

3.3.10.1 Execution Trace Capture (PMC_{42}.mode='000)

Section 3.3.10.1.1 through Section 3.3.10.1.3 describe the operation of the Execution Trace Buffer when configured to capture an execution trace (or "enhanced" branch trace).

3.3.10.1.1 Execution Trace Buffer Collection Conditions

The execution trace buffer configuration register (PMC_{39}) defines the conditions under which execution flow changes are to be captured. These conditions are given in Figure 3-23 and Table 3-26, which refer to conditions associated with the branch prediction. These conditions are:

- Whether the target of the branch should be captured
- The path of the branch (not taken/taken), and
- Whether or not the branch path was mispredicted
- Whether or not the target of the branch was mispredicted
- What type of branch should be captured

Note: All instructions eligible for capture are subject to filtering by the "plm" field but only branches are affected by PMC_{39}'s other filters (tm,ptm,ppm and brt) as well as the Instruction Addr Range and Opcode Match filters.

Figure 3-23. Execution Trace Buffer Configuration Register (PMC_{39})

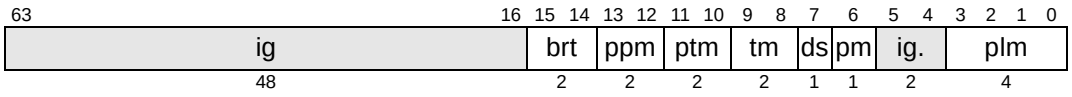


Table 3-26. Execution Trace Buffer Configuration Register Fields (PMC_{39})

Field	Bits	Description
plm	3:0	See Table 3-5 Note: This mask is applied at the time the event's source address is captured. Once the source IP is captured, the target IP of this event is always captured even if the ETB is disabled.
ig	5:4	Reads zero; writes are ignored
pm	6	See Table 3-5 Note: This bit is applied at the time the event's source address is captured. Once the source IP is captured, the target IP of this event is always captured even if the ETB is disabled.
ds	7	Data selector: 1: reserved (undefined data is captured in lieu of the target address) 0: capture branch target
tm	9:8	Taken Mask: 11: all Itanium® instruction branches 10: Taken Itanium instruction branches only 01: Not Taken Itanium instruction branches only 00: No branch is captured
ptm	11:10	Predicted Target Address Mask: 11: capture branch regardless of target prediction outcome 10: branch target address predicted correctly 01: branch target address mispredicted 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
brt	15:14	Branch Type Mask: 11: only non-return indirect branches captured 10: only return branches will be captured 01: only IP-relative branches will be captured 00: all branches are captured
ig	63:16	Reads zero; writes are ignored

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

```
(not PSR.is)
    and (       (tm[1] - branch taken)
            or (tm[0] - branch not taken)
        )
    and (       (ptm[1] - hardware predicted target address correctly)
            or  (ptm[0] - hardware mispredicted target address)
        )
    and (       (ppm[1] - hardware predicted the branch path correctly)
            or  (ppm[0] - hardware mispredicted the branch path)
        )
    and (       not (not ptm[1] and ptm[0] and not ppm[1] and ppm[0] )
        )           - hardware mispredicted path AND target
    and (       not ds
        )
```

To capture all correctly predicted Itanium instruction branches, the Montecito execution trace buffer configuration settings in PMC_{39} should be: ds=0, tm=11, ptm=10, ppm=10,brt=00.

Either branches whose path was mispredicted can be captured (ds=0, tm=11, ptm=11, ppm=01,brt=00) or branches with a target misprediction (ds=0, tm=11, ptm=01, ppm=11,brt=00) can be captured but not both. A setting of ds=0, tm=11, ptm=01, ppm=01,brt=00 will result in an empty buffer. If a branch's path is mispredicted, no target prediction is recorded.

Instruction Address Range Matching (Section 3.3.5) and Opcode Matching (Section 3.3.5) may also be used to constrain what is captured in the execution trace buffer.

3.3.10.1.2 Execution Trace Buffer Data Format (PMC_{42}.mode='000)

Figure 3-24. Execution Trace Buffer Register Format (PMD_{48-63}, where PMC_{39}.ds == 0)

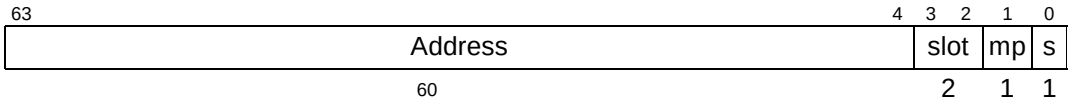


Table 3-27. Execution Trace Buffer Register Fields (PMD_{48-63}) (PMC_{42}.mode='000)

Field	Bit Range	Description
s	0	Source Bit 1: contents of register is the source address of a monitored event (branch, rfi, exception or failed chk) 0: contents of register is a target or undefined (if PMC_{39}.ds = 1)
mp	1	Mispredict Bit if s=1 and mp=1: mispredicted event (e.g. target, predicate or back end misprediction) if s=1 and mp=0: correctly predicted event if s=0 and mp=1: valid target address if s=0 and mp=0: invalid ETB register rfi/exceptions/failed_chk are all considered as mispredicted events and are encoded as above.
slot	3:2	if s=0: undefined if s=1: Slot index of first taken event in bundle 00: Itanium processor Slot 0 source/target 01: Itanium processor Slot 1 source/target 10: Itanium processor Slot 2 source/target 11: this was a not taken event
Address	63:4	if s=1: 60-bit bundle address of Itanium instruction branch if ds=0 and s=0: 60-bit target bundle address of Itanium instruction branch

The sixteen execution trace buffer registers PMD_{48-63} provide information about the outcome of a captured event sequence. The branch trace buffer registers (PMD_{48-63}) contain valid data only when event collection is frozen (PMC_0.fr is one). While event collection is enabled, reads of PMD_{48-63} return undefined values. The registers follow the layout defined in Figure 3-24, and Table 3-27 contain the address of either a captured branch instruction (s-bit=1) or a branch target (s-bit=0). For branch instructions, the mp-bit indicates a branch misprediction. An execution trace register with a zero s-bit and a zero mp-bit indicates an invalid buffer entry. The slot field captures the slot number of the first taken Itanium instruction branch in the captured instruction bundle. A slot number of 3 indicates a not-taken branch.

In every cycle in which a qualified Itanium instruction branch retires[1], its source bundle address and slot number are written to the branch trace buffer. If within the next clock, the target instruction bundle contains a branch that retires and meets the same conditions, the address of the second

branch is stored. Otherwise, either the branches' target address (PMC_{39}.ds=0) or details of the branch prediction (PCM_{39}.ds=1) are written to the next buffer location. As a result, the execution trace buffer may contain a mixed sequence of the branches and targets.

The Montecito branch trace buffer is a circular buffer containing the last four to eight qualified Itanium instruction branches. The Execution Trace Buffer Index Register (PMD_{38}) defined in Figure 3-25 and Table 3-28 identify the most recently recorded branch or target. In every cycle in which a qualified branch or target is recorded, the execution buffer index (ebi) 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 ebi field of PMD_{38} 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_{38}:

$$\text{last-written-PMD-index} = 48 + ([\ 16 * PMD_{38}.\text{full}) + (PMC_{38}.\text{ebi} - 1)] \ \%\ 16)$$

If both the full bit and the ebi field of PMD_{38} 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_{63} to PMD_{48}. Once set, the full bit remains set until explicitly cleared by software, i.e. it is a sticky bit. Software can reset the ebi index and the full bit by writing to PMD_{38}.

PMD_{39} provides additional information related to the ETB entries.

Figure 3-25. Execution Trace Buffer Index Register Format (PMD_{38})

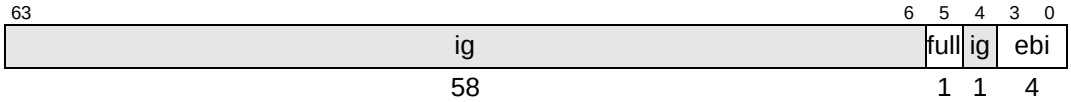
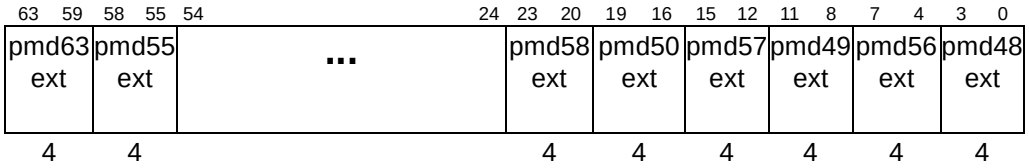


Table 3-28. Execution Trace Buffer Index Register Fields (PMD_{38})

Field	Bit Range	Description
ebi	3:0	Execution Buffer Index [Range 0..15 - Index 0 indicates PMD_{48}] Pointer to the next execution 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
ig	4	Reads zero; Writes are ignored
full	5	Full Bit (sticky) if full=1: execution trace buffer has wrapped if full=0: execution trace buffer has not wrapped
ig	63:6	Reads zero; Writes are ignored

Figure 3-26. Execution Trace Buffer Extension Register Format (PMD_{39}) (PMC_{42}.mode='1xx)



1. In some cases, the Montecito processor execution 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 multi-way branches.

Table 3-29. Execution Trace Buffer Extension Register Fields (PMD_{39}) (PMC_{42}.mode='1xx)

Field	Bit Range	Bits	Description
pmd48 ext	3:0	3:2	**ignored** Reads zero; writes are ignored
		1	**bruflush** If PMD48.bits[1:0] = 11, 1 = back end mispredicted the branch and the pipeline was flushed by it 0 = no pipeline flushes are associated with this branch
		0	**b1** if PMD48.s = 1, then 1 = branch was from bundle 1, add 0x1 to PMD48.bits[63:4] 0 = branch was from bundle 0, no correction is necessary else, ignore
pmd56 ext	7:4		Same as above for PMD56
pmd49 ext	11:8		Same as above for PMD49
pmd57 ext	15:12		Same as above for PMD57
pmd50 ext	19:16		Same as above for PMD50
pmd58 ext	23:20		Same as above for PMD58
so on	so on		so on
pmd63 ext	63:60		Same as above for PMD63

3.3.10.1.3 Notes on the Execution Trace Buffer

Although the Montecito ETB does not capture asynchronous interrupts as events, the address of these handlers can be captured as target addresses. This could happen if, at the target of a captured event (e.g. taken branch), an asynchronous event is taken before executing any instruction at the target.

3.3.10.2 IP Event Address Capture (PMC_{42}.mode='1xx)

Montecito has a new feature called Instruction Pointer Address Capture (or IP-EAR). This feature is intended to facilitate the correlation of performance monitoring events to IP values. To do this, the Montecito's Execution Trace Buffer (ETB) can be configured to capture IPs of retired instructions. When a performance monitoring event is used to trigger an IP-EAR freeze, if the IP which caused the event gets to retirement there is a good chance that IP would be captured in the ETB. The IP-EAR freezes after a programmable number of cycles following a PMU freeze as described below

Register PMC_{42} is used to configure this feature and the ETB registers($PMD_{48\text{-}63,39}$) are used to capture the data. PMD_{38} holds the index and overflow bits for the IP Buffer much as it does for the ETB.

Note: Setting PMC_{42}.mode to a non-0 value will override the setting of PMC_{39} (the configuration register for the normal BTB mode).

Figure 3-27. IP-EAR Configuration Register (PMC_{42})

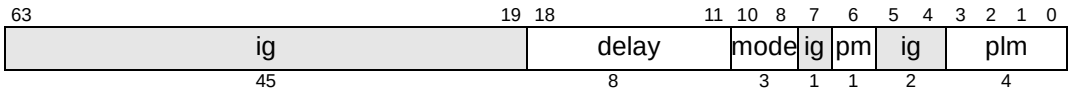


Table 3-30. IP-EAR Configuration Register Fields (PMC_{42})

Field	Bits	Description
plm	3:0	See Table 3-5, “Performance Monitor PMC Register Control Fields (PMC4-15)”
ig	5:4	Reads zero; Writes are ignored
pm	6	See Table 3-5, “Performance Monitor PMC Register Control Fields (PMC4-15)”
ig	7	Reads zero; Writes are ignored
mode	10:8	IP EAR mode: 000: ETB Mode (IP-EAR not functional; ETB is functional) 100: IP-EAR Mode (IP-EAR is functional; ETB not functional)
delay	18:11	Programmable delay before freezing
ig	63:20	Reads zero; Writes are ignored

The IP_EAR functions by continuously capturing retired IPs in PMD_{48-63} as long as it is enabled. It captures retired IPs and the elapsed time between retirements. Up to 16 entries can be captured.

The IP-EAR has a slightly different freezing model than the rest of the Performance Monitors. It is capable of delaying its freeze for a number of cycles past the point of PMU freeze. The user can program an 8-bit number to determine the number of cycles the freeze will be delayed.

Note: PMD_{48-63} are not, in fact, 68b registers. Figure 3-28 and Figure 3-29 represent the virtual layout of an execution trace buffer entry in IP-EAR mode for the sake of clarity. The higher order bits [67-64] for each entry are mapped into PMD_{39} as described in Table 3-33.

Figure 3-28. IP-EAR data format (PMD_{48-63}, where PMC_{42}.mode == 100 and PMD_{48-63}.ef =0)

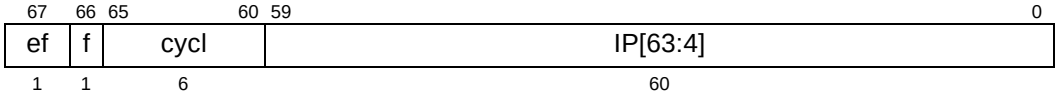


Figure 3-29. IP-EAR data format (PMD_{48-63}, where PMC_{42}.mode == 100 and PMD48-63.ef =1)

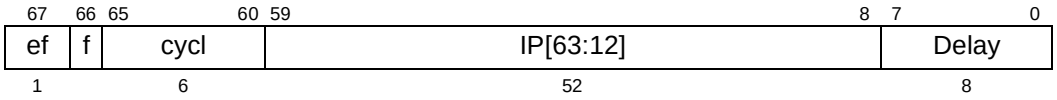


Table 3-31. IP-EAR Data Register Fields (PMD_{48-63}) (Sheet 1 of 2)(PMC_{42}.mode=‘1xx)

Field	Bits	Description
cycl	63:60	**Elapsed cycles** 4-bit least significant bits of a 6-bit elapsed cycle count from the previous retired IP. This is a saturating counter and would stay at all 1s when counted up to the maximum value. Note: the 2 most significant bits for each entry are found in PMD_{39}. See below.

Table 3-31. IP-EAR Data Register Fields (PMD_{48-63}) (Sheet 2 of 2)(PMC_{42}.mode='1xx)

Field	Bits	Description
IP	59:8	**Retired IP value**; bits[63:12]
delay	7:0	**Delay count** **If ef = 1** **Indicates the remainder of the delay count** **Else** Retired IP value**: bits[11:4]**

Figure 3-30. IP Trace Buffer Index Register Format (PMD_{38})(PMC_{42}.mode='1xx)

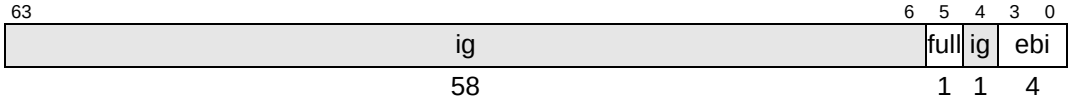


Table 3-32. IP Trace Buffer Index Register Fields (PMD_{38}) (PMC_{42}.mode='1xx)

Field	Bit Range	Description
ebi	3:0	IP Trace Buffer Index [Range 0..15 - Index 0 indicates PMD_{48}] Pointer to the next IP trace buffer entry to be written if full=1: points to the oldest recorded IP entry if full=0: points to the next location to be written
ig	4	Reads zero; Writes are ignored
full	5	Full Bit (sticky) if full=1: IP trace buffer has wrapped if full=0: IP trace buffer has not wrapped
ig	63:6	Reads zero; Writes are ignored

Figure 3-31. IP Trace Buffer Extension Register Format (PMD_{39}) (PMC_{42}.mode='1xx)

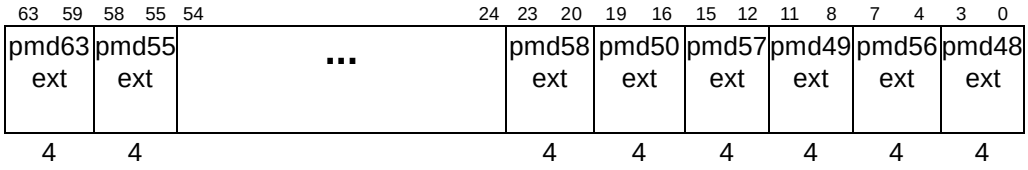


Table 3-33. IP Trace Buffer Extension Register Fields (PMD_{39}) (PMC_{42}.mode='1xx)

Field	Bit Range	Bits	Description
pmd48 ext	3:0	3:2	cycl **- Elapsed cycles** 2-bit most significant bits of a 6-bit elapsed cycle count from the previous retired IP. This is a saturating counter and would stay at all 1s when counted up to the maximum value.
		1	**f** - Flush Indicates whether there has been a pipe flush since the last entry
		0	**ef** - Early freeze if 1: The current entry is an early freeze case Early freeze occurs if: PSR bits causes IP-EAR to become disabled Thread switch
pmd56 ext	7:4		Same as above for PMD56
pmd49 ext	11:8		Same as above for PMD49
pmd57 ext	15:12		Same as above for PMD57
pmd50 ext	19:16		Same as above for PMD50
pmd58 ext	23:20		Same as above for PMD58
so on	so on		so on
pmd63 ext	63:60		Same as above for PMD63

3.3.10.2.1 Notes on the IP-EAR

When the IP-EAR freezes due to its normal freeze mechanism (i.e. PMU freeze + delay), it captures one last entry with "ef"=0. The IP value in this entry could be incorrect since there is no guarantee that the CPU would be retiring an IP at this particular time. Since this is always the youngest entry captured in IP_EAR buffer, it should be easier to identify this event.

3.3.11 Interrupts

As mentioned in Table 3-6, each one of registers PMD_{4-15} will cause an interrupt if the following conditions are all true:

- PMC_i.oi=1 (i.e. overflow interrupt is enabled for PMD_i) and PMD_i overflows. Note that there is only one interrupt line that will be raised regardless of which PMC/PMD set meets this condition.

This interrupt is an "External Interrupt" with Vector= 0x3000 and will be recognized only if the following conditions are true:

- PMV.m=0 and PMV.vector is set up correctly; i.e. Performance Monitor interrupts are not masked and a proper vector is programmed for this interrupt by executing a "`mov cr73=r2`".
- PSR.i =1 and PSR.ic=1; i.e. interruptions are unmasked and interruption collection is enabled in the Processor Status Register by executing either the "`ssm imm`" or "`mov psr.l=r2`" instruction.
- TPR.mmi=0 (i.e. all external interrupts are not masked) and TPR.mic is a value that the priority class that Performance Monitor Interrupt belongs to are not masked. For example if we assign vector 0xD2 to the Performance Monitor Interrupt, according to Table 5-7 "Interrupt Priorities, Enabling, and Masking" in Volume 2 of the *Intel® Itanium® Architecture Software Developer's Manual*, it will be priority class 13. So any value less than 13 for TPR.mic is okay for recognizing this interrupt. A "`mov cr66=r1`" will write to this register.

- There are no higher priority faults, traps, or external interrupts pending.

Interrupt Service routine needs to read IVR register "`mov r1=cr65`" in order to figure out the highest priority external interrupt which needs to be serviced.

Before returning from interrupt service routine, the Performance Monitor needs to be initialized such that the interrupt will be cleared. This could be done by clearing the PMC.oi and/or re-initializing the PMD which caused the interrupt (you will know this by reading PMC_0). In addition to this, all bits of PMC_0 need to be cleared if further monitoring needs to be done.

3.3.12 Processor Reset, PAL Calls, and Low Power State

Processor Reset: On processor hardware reset bits `oi` and `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:

- $PMC_{32,33,34,35}$ = 0xffffffffffffffff, (match all opcodes)
- PMC_{41} = 0x2078fefefefe, (no memory pipeline event constraints)
- PMC_{38} = 0xdb6, (no instruction address range constraints)
- PMC_{36} = 0xfffffff0, (no opcode match constraints)

All other performance monitoring related state is undefined.

Table 3-34. Information Returned by PAL_PERF_MON_INFO for the Montecito Processor

PAL_PERF_MON_INFO Return Value	Description	Montecito Processor Specific Value
PAL_RETIRED	8-bit unsigned event type for counting the number of untagged retired Itanium 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	48
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 Itanium instructions (event defined by PAL_RETIRED)	0xF0

PAL Call: As defined in the Volume 2 of the *Intel® Itanium® Architecture Software Developer's Manual*, the PAL call PAL_PERF_MON_INFO provides software with information about the implemented performance monitors. The Montecito processor specific values are summarized in Table 3-34.

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.

As a result, bus events occurring during lower power state (e.g. snoops) will not be counted. PAL_LIGHT_HALT preserves the original value of the PMC_0 register.

§

4 Performance Monitor Events

4.1 Introduction

This chapter describes the architectural and microarchitectural events measurable on the Montecito processor through the performance monitoring mechanisms described earlier in Chapter 3. The early sections of this chapter provide a categorized high-level view of the event list, grouping logically related events together. Computation (either directly by a counter in hardware or indirectly as a “derived” event) of common performance metrics is also discussed. Each directly measurable event is then described in greater detail in the alphabetized list of all processor events in Chapter 4.

The Montecito processor is capable of monitoring numerous events. The majority of events can be selected as input to any of the $PMD_{4\text{-}15}$ by programming bit [15:8] of the corresponding PMC to the hexadecimal values shown in the “event code” field of the event list. Please refer to Section 4.8.2 and Section 4.8.4 for events that have more specific requirements.

4.2 Categorization of Events

Performance related events are grouped into the following categories:

- Basic Events: clock cycles, retired instructions (Section 4.3)
- Instruction Dispersal Events: instruction decode and issue (Section 4.4)
- Instruction Execution Events: instruction execution, data and control speculation, and memory operations (Section 4.5)
- Stall Events: stall and execution cycle breakdowns (Section 4.6)
- Branch Events: branch prediction (Section 4.7)
- Memory Hierarchy: instruction and data caches (Section 4.8)
- System Events: operating system monitors (Section 4.9)
- TLB Events: instruction and data TLBs (Section 4.10)
- System Bus Events: (Section 4.11)
- RSE Events: Register Stack Engine (Section 4.12)
- Hyper-Threading Events (Section 4.13)

Each section listed above includes a table providing information on directly measurable events. The section may also contain a second table of events that can be derived from those that are directly measurable. These derived events may simply rename existing events or present steps to determine the value of common performance metrics. Derived events are not, however, discussed in the systematic event listing in Section 4.15.

Directly measurable events often use the PMC.umask field (See Chapter 3) to measure a certain variant of the event in question. Symbolic event names for such events include a period to indicate use of the umask, specified by four bits in the detailed event description (x’s are for don’t-cares).

The summary tables in the subsequent sections define events by specifying the following attributes:

- Symbol Name - Symbolic name used to denote this event.
- Event Code - Hexadecimal value to program into bits [15:8] of the appropriate PMC register in order to measure this event.
- IAR - Can this event be constrained by the Instruction Address Range registers?
- DAR - Can this event be constrained by the Data Address Range registers?
- OPC - Can this event by constrained by the Opcode Match registers?
- Max Inc/Cyc - Maximum Increment Per Cycle or the maximum value this event may be increased by each cycle.
- T - Type; Either A for Active, F for Floating, S for Self Floating or C for Causal (check table Table 4-42 for this information).
- Description - Brief description of the event.

4.2.1 Hyper-Threading and Event Types

The Montecito Processor implements a type of hardware based multi-threading that effectively allows two threads to coexist within a processor core although only one thread is “active” within the core’s pipeline at any moment in time. This affects how events are generated. Certain events may be generated after the thread they belong has become inactive. This also affects how events are assigned to the threads occupying the same core, which is also dependent upon which PMD the event was programmed into (see Section 3.3.2 for more information). Certain events do not have the concept of a “home” thread.

These effects are further complicated by the use of the “*.all*” field, which allows a user to choose to monitor a particular event for the thread being programmed to or for both threads (see Table 3-6). It should be noted that monitoring with *.all* enabled does not always produce valid results and in certain cases the setting of *.all* is ignored. Please refer to the individual events for further information.

To help decipher these effects, events have been classified by the following types:

- **Active** - this event can only occur when the thread that generated it is “active” (currently executing in the processor core’s pipeline) and is considered to be generated by the active thread. Either type of monitor can be used if *.all* is not set. Example(s): BE_EXE_BUBBLE and IA64_INST_RETIRED.
- **Causal** - this event does not belong to a thread. It is assigned to the active thread. Although it seems natural to use either type of monitor if *.all* is not set, due to implementation constraints, causal events should **only** be monitored in duplicated counters. There is one exception to this rule: CPU_OP_CYCLES can be measured in both types of counters. Example(s): CPU_OP_CYCLES and L2I_SNOOP_HITS.
- **Floating** - this event belongs to a thread, but could have been generated when its thread was inactive (or “in the background”). These events should **only** be monitored in duplicated counters. If *.all* is not set, only events associated with the monitoring thread will be captured. If *.all* is set, events associated with both threads will be captured during the time the monitoring thread has been assigned to a processor by the OS. Example(s): L2D_REFERENCES and ER_MEM_READ_OUT_LO.

- **Self Floating** - this is a hybrid event used to better categorize certain BUS and SI (System Interface) events. If this event was monitored with the .SELF umask, it is a Floating event. If any other umask is used it is considered Causal. These events should **only** be monitored in duplicated counters. Example(s): BUS_IO and SI_WRITEQ_INSERTS.

4.3 Basic Events

Table 4-1 summarizes two basic execution monitors. The Montecito retired instruction count, IA64_INST_RETIRED, includes both predicated true and predicated off instructions and `nop` instructions, but excludes RSE operations.

Table 4-1. Performance Monitors for Basic Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	T	Description
CPU_OP_CYCLES	0x12	Y	N	Y	1	C	CPU Operating Cycles
IA64_INST_RETIRED	0x08	Y	N	Y	6	A	Retired Itanium® Instructions

Table 4-2. Derived Monitors for Basic Events

Symbol Name	Description	Equation
IA64_IPC	Average Number of Itanium® Instructions Per Cycle During Itanium architecture-based Code Sequences	IA64_INST_RETIRED / CPU_OP_CYCLES

4.4 Instruction Dispersal Events

Instruction cache lines are delivered to the execution core and dispersed to the Montecito processor functional units. The Montecito processor can issue, or disperse, 6 instructions per clock cycle. In other words, the Montecito processor can issue to 6 instruction slots (or syllables).The following events are intended to give users an idea of how effectively instructions are dispersed and why they are not dispersed at full capacity. There are five reasons for not dispersing at full capacity. One is measured by DISP_STALLED. For every clock that dispersal is stalled, dispersal takes a hit of 6-syllables. The other four reasons are measured by SYLL_NOT_DISPERSED. Due to the way the hardware is designed, SYLL_NOT_DISPERSED may contain an overcount due to implicit and explicit bits; although this number should be small, SYLL_OVERCOUNT will provide an accurate count for it.

The relationship between these events is as follows:

6*(CPU_OP_CYCLES-DISP_STALLED) = INST_DISPERSED + SYLL_NOT_DISPERSED.ALL - SYLL_OVERCOUNT.ALL

Table 4-3. Performance Monitors for Instruction Dispersal Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
DISP_STALLED	0x49	N	N	N	1	Number of cycles dispersal stalled
INST_DISPERSED	0x4d	Y	N	N	6	Syllables dispersed from REN to REG stage
SYLL_NOT_DISPERSED	0x4e	Y	N	N	5	Syllables not dispersed
SYLL_OVERCOUNT	0x4f	Y	N	N	2	Syllables overcounted

4.5 Instruction Execution Events

Retired instruction counts, IA64_TAGGED_INST_RETIRED and NOPS_RETIRED, are based on tag information specified by the address range check and opcode match facilities. A separate event, PREDICATE_SQUASHED_RETIRED, is provided to count predicated off instructions.

The FP monitors listed in the table capture dynamic information about pipeline flushes and flush-to-zero occurrences due to floating-point operations. The FP_OPS_RETIRED event counts the number of retired FP operations.

As Table 4-4 describes, monitors for control and data speculation capture dynamic run-time information: the number of failed `chk.s` instructions (INST_FAILED_CHKS_RETIRED.ALL), the number of advanced load checks and check loads (INST_CHKA_LDC_ALAT.ALL), and failed advanced load checks and check loads (INST_FAILED_CHKA_LDC_ALAT.ALL) as seen by the ALAT. The number of retired `chk.s` instructions is monitored by the IA64_TAGGED_INST_RETIRED event, given the appropriate opcode mask. Since the Montecito processor ALAT is updated by operations on mispredicted branch paths, the number of advanced load checks and check loads need an explicit event (INST_CHKA_LDC_ALAT.ALL).

Table 4-4. Performance Monitors for Instruction Execution Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
ALAT_CAPACITY_MISS	0x58	Y	Y	Y	2	ALAT Entry Replaced
FP_FAILED_FCHKF	0x06	Y	N	N	1	Failed fchkf
FP_FALSE_SIRSTALL	0x05	Y	N	N	1	SIR stall without a trap
FP_FLUSH_TO_ZERO	0x0b	Y	N	N	2	FP Result Flushed to Zero
FP_OPS_RETIRED	0x09	Y	N	N	6	Retired FP operations
FP_TRUE_SIRSTALL	0x03	Y	N	N	1	SIR stall asserted and leads to a trap
IA64_TAGGED_INST_RETIRED	0x08	Y	N	Y	6	Retired Tagged Instructions
INST_CHKA_LDC_ALAT	0x56	Y	Y	Y	2	Advanced Check Loads
INST_FAILED_CHKA_LDC_ALAT	0x57	Y	Y	Y	1	Failed Advanced Check Loads
INST_FAILED_CHKS_RETIRED	0x55	N	N	N	1	Failed Speculative Check Loads
LOADS_RETIRED	0xcd	Y	Y	Y	4	Retired Loads
MISALIGNED_LOADS_RETIRED	0xce	Y	Y	Y	4	Retired Misaligned Load Instructions
MISALIGNED_STORES_RETIRED	0xd2	Y	Y	Y	2	Retired Misaligned Store Instructions
NOPS_RETIRED	0x50	Y	N	Y	6	Retired NOP Instructions

Table 4-4. Performance Monitors for Instruction Execution Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
PREDICATE_SQUASHED_RETIRED	0x51	Y	N	Y	6	Instructions Squashed Due to Predicate Off
STORES_RETIRED	0xd1	Y	Y	Y	2	Retired Stores
UC_LOADS_RETIRED	0xcf	Y	Y	Y	4	Retired Uncacheable Loads
UC_STORES_RETIRED	0xd0	Y	Y	Y	2	Retired Uncacheable Stores

Table 4-5. Derived Monitors for Instruction Execution Events

Symbol Name	Description	Equation
ALAT_EAR_EVENTS	Counts the number of ALAT events captured by EAR	DATA_EAR_EVENTS
CTRL_SPEC_MISS_RATIO	Control Speculation Miss Ratio	INST_FAILED_CHKS_RETIRED.ALL / IA64_TAGGED_INST_RETIRED[chk.s]
DATA_SPEC_MISS_RATIO	Data Speculation Miss Ratio	INST_FAILED_CHKA_LDC_ALAT.ALL / INST_CHKA_LDC_ALAT.ALL

4.6 Stall Events

Montecito processor stall accounting is separated into front-end and back-end stall accounting. Back-end and front-end events should not be compared since they are counted in different stages of the pipeline.

The back-end can be stalled due to five distinct mechanisms: FPU/L1D, RSE, EXE, branch/exception or the front-end. BACK_END_BUBBLE provides an overview of which mechanisms are producing stalls while the other back-end counters provide more explicit information broken down by category. Each time there is a stall, a bubble is inserted in only one location in the pipeline. Each time there is a flush, bubbles are inserted in all locations in the pipeline. With the exception of BACK_END_BUBBLE, the back-end stall accounting events are prioritized in order to mimic the operation of the main pipe (i.e. priority form high to low is given to: BE_FLUSH_BUBBLE.XPN, BE_FLUSH_BUBBLE.BRU, L1D_FPU stalls, EXE stalls, RSE stalls, front-end stalls). This prioritization guarantees that the events are mutually exclusive and only the most important cause, the one latest in the pipeline, is counted.

The Montecito processor's front-end can be stalled due to seven distinct mechanisms: FEFLUSH, TLBMISS, IMISS, branch, FILL-RECIRC, BUBBLE, IBFULL (listed in priority from high to low). The front-end stalls have exactly the same effect on the pipeline so their accounting is simpler.

During every clock in which the CPU is not in a halted state, the back-end pipeline has either a bubble or it retires 1 or more instructions, CPU_OP_CYCLES = BACK_END_BUBBLE.all + (IA64_INST_RETIRED >= 1). To further investigate bubbles occurring in the back-end of the pipeline the following equation holds true: BACK_END_BUBBLE.all = BE_RSE_BUBBLE.all + BE_EXE_BUBBLE.all + BE_L1D_FPU_BUBBLE.all + BE_FLUSH_BUBBLE.all + BACK_END_BUBBLE.fe.

Note: CPU_OP_CYCLES is not incremented during a HALT state. If a measurement is set up to match clock cycles to bubbles to instructions retired (as outlined above) and a halt occurs within the measurement interval, measuring CYCLES_HALTED in PMD_{10} may be used to compensate.

Each of the stall events (summarized in Table 4-6) take a umask to choose among several available sub-events. Please refer to the detailed event descriptions in Section 4.15 for a list of available sub-events and their individual descriptions.

Table 4-6. Performance Monitors for Stall Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
BACK_END_BUBBLE	0x00	N	N	N	1	Full pipe bubbles in main pipe
BE_EXE_BUBBLE	0x02	N	N	N	1	Full pipe bubbles in main pipe due to Execution unit stalls
BE_FLUSH_BUBBLE	0x04	N	N	N	1	Full pipe bubbles in main pipe due to flushes
BE_L1D_FPU_BUBBLE	0xca	N	N	N	1	Full pipe bubbles in main pipe due to FP or L1D cache
BE_LOST_BW_DUE_TO_FE	0x72	N	N	N	2	Invalid bundles if BE not stalled for other reasons
BE_RSE_BUBBLE	0x01	N	N	N	1	Full pipe bubbles in main pipe due to RSE stalls
FE_BUBBLE	0x71	N	N	N	1	Bubbles seen by FE
FE_LOST_BW	0x70	N	N	N	2	Invalid bundles at the entrance to IB
IDEAL_BE_LOST_BW_DUE_TO_FE	0x73	N	N	N	2	Invalid bundles at the exit from IB

4.7 Branch Events

Note that for branch events, retirement means a branch was reached and committed regardless of its predicate value. Details concerning prediction results are contained in pairs of monitors. For accurate misprediction counts, the following measurement must be taken:

BR_MISPRED_DETAIL.[umask] - BR_MISPRED_DETAIL2.[umask]

By performing this calculation for every umask, one can obtain a true value for the BR_MISPRED_DETAIL event.

The method for obtaining the true value of BR_PATH_PRED is slightly different. When there is more than one branch in a bundle and one is predicted as taken, all the higher number ports are forced to a predicted not taken mode without actually knowing the their true prediction.

The true OKPRED_NOTTAKEN predicted path information can be obtained by calculating:

BR_PATH_PRED.[branch type].OKPRED_NOTTAKEN - BR_PATH_PRED2.[branch type].UNKNOWNPRED_NOTTAKEN using the same “branch type” (ALL, IPREL, RETURN, NRETIND) specified for both events.

Similarly, the true MISPRED_TAKEN predicted path information can be obtained by calculating:

BR_PATH_PRED.[branch type].MISPRED_TAKEN - BR_PATH_PRED2.[branch type].UKNOWNPRED_TAKEN using the same “branch type” (ALL, IPREL, RETURN, NRETIND) selected for both events.

BRANCH_EVENT counts the number of events captured by the Execution Trace Buffer. For detailed information on the ETB please refer to Section 3.3.10.

Table 4-7. Performance Monitors for Branch Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
BE_BR_MISPRED_DETAIL	0x61	Y	N	Y	1	BE branch misprediction detail
BRANCH_EVENT	0x11	Y	N	Y	1	Branch Event Captured
BR_MISPRED_DETAIL	0x5b	Y	N	Y	3	FE Branch Mispredict Detail
BR_MISPRED_DETAIL2	0x68	Y	N	Y	2	FE Branch Mispredict Detail (Unknown path component)
BR_PATH_PRED	0x54	Y	N	Y	3	FE Branch Path Prediction Detail
BR_PATH_PRED2	0x6a	Y	N	Y	2	FE Branch Path Prediction Detail (Unknown prediction component)
ENCBR_MISPRED_DETAIL	0x63	Y	N	Y	3	Number of encoded branches retired

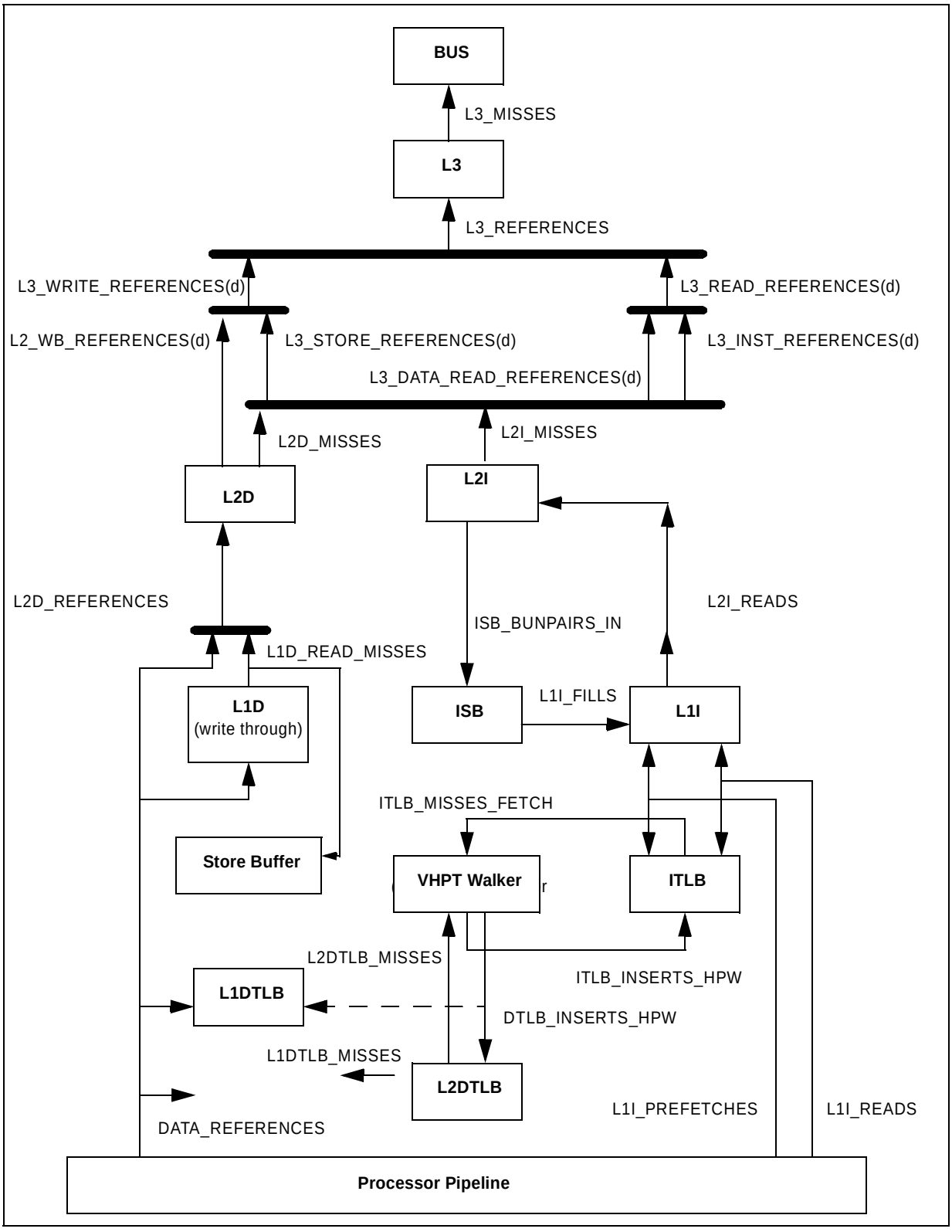
4.8 Memory Hierarchy

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

- L1 Instruction Cache and Prefetch Events (Section 4.8.1)
- L1 Data Cache Events (Section 4.8.2)
- L2 Instruction Cache Events (Section 4.8.3)
- L2 Data Cache Events (Section 4.8.4)
- L3 Cache Events (Section 4.8.5)

An overview of the Montecito processor’s three level memory hierarchy and its event monitors is shown in Figure 4-1. The instruction and the data stream work through separate L1 caches. The L1 data cache is a write-through cache. Two separate L2I and L2D caches serve both the L1 instruction and data caches respectively, and are backed by a large unified L3 cache. Events for individual levels of the cache hierarchy are described in the Section 4.8.1 through Section 4.8.3.

Figure 4-1. Event Monitors in the Itanium® 2 Processor Memory Hierarchy



4.8.1 L1 Instruction Cache and Prefetch Events

Table 4-8 describes and summarizes the events that the Montecito processor provides to monitor L1 instruction cache demand fetch and prefetch activity. The instruction fetch monitors distinguish between demand fetch (L1I_READS) and prefetch activity (L1I_PREFETCHES). The amount of data returned from the L2I to the L1 instruction cache and the Instruction Streaming Buffer is monitored by two events, L1I_FILLS and ISB_LINES_IN. The L1I_EAR_EVENTS monitor counts how many instruction cache or L1ITLB misses are captured by the instruction event address register.

The L1 instruction cache and prefetch events can be qualified by the instruction address range check, but not by the opcode matcher. Since instruction cache and prefetch events occur early in the processor pipeline, they include events caused by speculative, wrong-path instructions as well as predicated-off instructions. Since the address range check is based on speculative instruction addresses rather than retired 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 Chapter 3 for details).

L1I_EAR_EVENTS counts the number of events captured by the Montecito processor's instruction EARs. Please refer to Chapter 3 for more detailed information about the instruction EARs.

Table 4-8. Performance Monitors for L1/L2 Instruction Cache and Prefetch Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
ISB_BUNPAIRS_IN	0x46	Y	N	N	1	Bundle pairs written from L2I into FE
L1I_EAR_EVENTS	0x43	Y	N	N	1	Instruction EAR Events
L1I_FETCH_ISB_HIT	0x66	Y	N	N	1	"Just-in-time" instruction fetch hitting in and being bypassed from ISB
L1I_FETCH_RAB_HIT	0x65	Y	N	N	1	Instruction fetch hitting in RAB
L1I_FILLS	0x41	Y	N	N	1	L1 Instruction Cache Fills
L1I_PREFETCHES	0x44	Y	N	N	1	L1 Instruction Prefetch Requests
L1I_PREFETCH_STALL	0x67	N	N	N	1	Why prefetch pipeline is stalled?
L1I_PURGE	0x4b	Y	N	N	1	L1ITLB purges handled by L1I
L1I_PVAB_OVERFLOW	0x69	N	N	N	1	PVAB overflow
L1I_RAB_ALMOST_FULL	0x64	N	N	N	1	Is RAB almost full?
L1I_RAB_FULL	0x60	N	N	N	1	Is RAB full?
L1I_READS	0x40	Y	N	N	1	L1 Instruction Cache Reads
L1I_SNOOP	0x4a	Y	Y	Y	1	Snoop requests handled by L1I
L1I_STRM_PREFETCHES	0x5f	Y	N	N	1	L1 Instruction Cache line prefetch requests
L2I_DEMAND_READS	0x42	Y	N	N	1	L1 Instruction Cache and ISB Misses
L2I_PREFETCHES	0x45	Y	N	N	1	L2 Instruction Prefetch Requests

Table 4-9. Derived Monitors for L1 Instruction Cache and Prefetch Events

Symbol Name	Description	Equation
L1I_MISSES	L1I Misses	L2I_DEMAND_READS
ISB_LINES_IN	Number of cache lines written from L2I (and beyond) into the front end	ISB_BUNPAIRS_IN/4
L1I_DEMAND_MISS_RATIO	L1I Demand Miss Ratio	L2I_DEMAND_READS / L1I_READS
L1I_MISS_RATIO	L1I Miss Ratio	(L1I_MISSES + L2I_PREFETCHES) / (L1I_READS + L1I_PREFETCHES)
L1I_PREFETCH_MISS_RATIO	L1I Prefetch Miss Ratio	L2I_PREFETCHES / L1I_PREFETCHES
L1I_REFERENCES	Number of L1 Instruction Cache reads and fills	L1I_READS + L1I_PREFETCHES

4.8.2 L1 Data Cache Events

Table 4-10 lists the Montecito processor's L1 data cache monitors. As shown in Figure 4-1, the write-through L1 data cache services cacheable loads, integer and RSE loads, check loads and hinted L2 memory references. DATA_REFERENCES is the number of issued data memory references.

L1 data cache reads (L1D_READS) and L1 data cache misses (L1D_READ_MISSES) monitor the read hit/miss rate of the L1 data cache. RSE operations are included in all data cache monitors, but are not broken down explicitly. The DATA_EAR_EVENTS monitor counts how many data cache or DTLB misses are captured by the Data Event Address Register. Please refer to Section 3.3.9 for more detailed information about the data EARs.

L1D cache events have been divided into 6 sets (sets 0,1,2,3,4,6; set 5 is reserved). Events from different sets of L1D Cache events cannot be measured at the same time. Each set is selected by the event code programmed into PMC5 (i.e. if you want to measure any of the events in this set, one of them needs to be measured by PMD5). There are no limitations on umasks. Monitors belonging to each set are explicitly presented in Table 4-10 through Table 4-16.

Table 4-10. Performance Monitors for L1 Data Cache Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
DATA_EAR_EVENTS	0xc8	Y	Y	Y	1	L1 Data Cache EAR Events
L1D_READS_SET0	0xc2	Y	Y	Y	2	L1 Data Cache Reads
DATA_REFERENCES_SET0	0xc3	Y	Y	Y	4	Data memory references issued to memory pipeline
L1D_READS_SET1	0xc4	Y	Y	Y	2	L1 Data Cache Reads
DATA_REFERENCES_SET1	0xc5	Y	Y	Y	4	Data memory references issued to memory pipeline
L1D_READ_MISSES	0xc7	Y	Y	Y	2	L1 Data Cache Read Misses

4.8.2.1 L1D Cache Events (set 0)

Table 4-11. Performance Monitors for L1D Cache Set 0

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L1DTLB_TRANSFER	0xc0	Y	Y	Y	1	L1DTLB misses hit in L2DTLB for access counted in L1D_READS
L2DTLB_MISSES	0xc1	Y	Y	Y	4	L2DTLB Misses
L1D_READS_SET0	0xc2	Y	Y	Y	2	L1 Data Cache Reads
DATA_REFERENCES_SET0	0xc3	Y	Y	Y	4	Data memory references issued to memory pipeline

4.8.2.2 L1D Cache Events (set 1)

Table 4-12. Performance Monitors for L1D Cache Set 1

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L1D_READS_SET1	0xc4	Y	Y	Y	2	L1 Data Cache Reads
DATA_REFERENCES_SET1	0xc5	Y	Y	Y	4	Data memory references issued to memory pipeline
L1D_READ_MISSES	0xc7	Y	Y	Y	2	L1 Data Cache Read Misses

4.8.2.3 L1D Cache Events (set 2)

Table 4-13. Performance Monitors for L1D Cache Set 2

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
BE_L1D_FPU_BUBBLE	0xca	N	N	N	1	Full pipe bubbles in main pipe due to FP or L1D cache

4.8.2.4 L1D Cache Events (set 3)

Table 4-14. Performance Monitors for L1D Cache Set 3

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
LOADS_RETIRED	0xcd	Y	Y	Y	4	Retired Loads
MISALIGNED_LOADS_RETIRED	0xce	Y	Y	Y	4	Retired Misaligned Load Instructions
UC_LOADS_RETIRED	0xcf	Y	Y	Y	4	Retired Uncacheable Loads

4.8.2.5 L1D Cache Events (set 4)

Table 4-15. Performance Monitors for L1D Cache Set 4

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
MISALIGNED_STORES_RETIRED	0xd2	Y	Y	Y	2	Retired Misaligned Store Instructions
STORES_RETIRED	0xd1	Y	Y	Y	2	Retired Stores
UC_STORES_RETIRED	0xd0	Y	Y	Y	2	Retired Uncacheable Stores

4.8.2.6 L1D Cache Events (set 6)

Table 4-16. Performance Monitors for L1D Cache Set 6

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
LOADS_RETIRED_INTG	0xd8	Y	Y	Y	2	Integer loads retired
SPEC_LOADS_NATTED	0xd9	Y	Y	Y	2	Times ld.s or ld.sa NaT'd

4.8.3 L2 Instruction Cache Events

Table 4-17. Performance Monitors for L2I Cache

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2I_READS	0x78	Y	N	Y	1	L2I Cacheable Reads
L2I_UC_READS	0x79	Y	N	Y	1	L2I uncacheable reads
L2I_VICTIMIZATIONS	0x7a	Y	N	Y	1	L2I victimizations
L2I_RECIRCULATES	0x7b	Y	N	Y	1	L2I recirculates
L2I_L3_REJECTS	0x7c	Y	N	Y	1	L3 rejects
L2I_HIT_CONFLICTS	0x7d	Y	N	Y	1	L2I hit conflicts
L2I_SPEC_ABORTS	0x7e	Y	N	Y	1	L2I speculative aborts
L2I_SNOOP_HITS	0x7f	Y	N	Y	1	L2I snoop hits

Table 4-18. Derived Monitors for L2I Cache (Sheet 1 of 2)

Symbol Name	Description	Equation
L2I_SNOOPS	Number of snoops received by the L2I.	L1I_SNOOPS
L2I_FILLS	L2I Fills	L2I_READS.MISS.DMND + L2I_READS.MISS.PFTCH
L2I_FETCHES	Requests made to L2I due to demand instruction fetches.	L2I_READS.ALL.DMND
L2I_REFERENCES	Instructions requests made to L2I	L2I_READS.ALL.ALL

Table 4-18. Derived Monitors for L2I Cache (Sheet 2 of 2)

Symbol Name	Description	Equation
L2I_MISS_RATIO	Percentage of L2I Misses	L2I_READS.MISS/L2I_READS.ALL
L2I_HIT_RATIO	Percentage of L2I Hits	L2I_READS.HIT/L2I_READS.ALL

4.8.4 L2 Data Cache Events

Table 4-19 summarizes the events available to monitor the Montecito processor L2D cache.

Most L2D events have been divided into 8 sets. Only events within two of these sets (or non-L2D events) can be measured at the same time. These two sets are selected by the event code programmed into PMC4 and PMC6 (i.e. if you want to measure any of the events in a particular set, one of these events needs to be measured by PMD4 or PMD6).

Note: The opposite holds true. If PMC4 is not programmed to monitor an L2D event, yet PMC5 or PMC8 are (similarly with PMC6->PMC7/9), PMD values are undefined. Also note that

Any event set can be measured by programming either PMC4 or PMC6. Once PMC4 is programmed to measure an event from one L2D event set, PMD4, PMD5, and PMD8 can only measure events from the **same** L2D event set (PMD5,8 share the umask programmed into PMD4). Similarly, once PMC6 is programmed to monitor another set (could be the same set as measured by PMC4), PMD6, PMD7 and PMD9 can measure events from this set only. None of the L2 data cache events can be measured using PMD10-15.

Support for the .all bit has the same restrictions as the set restrictions. The value set for ".all" in PMC4 applies to both the L1D events selected by it. Hence, even though the ".all" values in PMC5 and PMC8 are different from the value in PMC4, PMC4's value selects the capability. This is same with PMC6,7,9. Original Montecito documentation claimed that Thread 0 PMC4 .me/.all applied to PMC4-PMC7 but that is no longer true. This bit is available for both the threads. Hence, it is possible for one thread's PMDs to monitor just the events credited for that thread while the other thread's PMDs can monitor events for both threads (if PMC4.all is set). Note that some events do not support .all counting. If .all counting is enabled for events that don't support it, the resulting counts will be wrong.

While the L2D events support threading, not all counts have access to exact thread id bit needed. Each count is labeled with one of ActiveTrue, ActiveApprox, or TrueThrd. ActiveTrue means that the event is counted with the current active thread, and that thread is the only thread that can see the event when it is counted. ActiveApprox means the event is counted with the current active thread, but there are some corner cases were the event may actually be due to the other non-Active thread. It is assumed in most cases that the error due this approximation will be negligible. TrueThrd indicates that the L2D cache has knowledge of what thread the count belongs to besides the active thread indication, and that knowledge is always correct.

Table 4-19. Performance Monitors for L2 Data Cache Events (Sheet 1 of 2)

Symbol Name	Event Code	IAR	DAR	OPC	.all capable	Max Inc/Cyc	Description
L2D_OZQ_CANCELS0	0xe0	Y	Y	Y	Y	4	L2D OZQ cancels
L2D_OZQ_FULL	0xe1 0xe3	N	N	N	N	1	L2D OZQ is full
L2D_OZQ_CANCELS1	0xe2	Y	Y	Y	Y	4	L2D OZQ cancels

Table 4-19. Performance Monitors for L2 Data Cache Events (Sheet 2 of 2)

Symbol Name	Event Code	IAR	DAR	OPC	.all capable	Max Inc/Cyc	Description
L2D_BYPASS	0xe4	Y	Y	Y	Y/N	1	L2D Hit or Miss Bypass (.all support is umask dependent)
L2D_OZQ_RELEASE	0xe5	N	N	N	N	1	Clocks with release ordering attribute existed in L2D OZQ
L2D_REFERENCES	0xe6	Y	Y	Y	Y	4	Data RD/WR access to L2D
L2D_L3ACCESS_CANCEL	0xe8	Y	Y	Y	N	1	Canceled L3 accesses
L2D_OZDB_FULL	0xe9	N	N	N	Y	1	L2D OZ data buffer is full
L2D_FORCE_RECIRC	0xea	Y	Y	Y	Y/N	4	Forced recirculates
L2D_ISSUED_RECIRC_OZQ_ACC	0xeb	Y	Y	Y	Y	1	Count the number of times a recirculate issue was attempted and not preempted
L2DBAD_LINES_SELECTED	0xec	Y	Y	Y	Y	4	Valid line replaced when invalid line is available
L2D_STORE_HIT_SHARED	0xed	Y	Y	Y	Y	2	Store hit a shared line
L2D_OZQ_ACQUIRE	0xef	N	N	N	Y	1	Clocks with acquire ordering attribute existed in L2D OZQ
L2D_OPS_ISSUED	0xf0	Y	Y	Y	N	4	Different operations issued by L2D
L2D_FILLB_FULL	0xf1	N	N	N	N	1	L2D Fill buffer is full
L2D_FILL_MESI_STATE	0xf2	Y	Y	Y	Y	1	MESI states of fills to L2D cache
L2D_VICTIMB_FULL	0xf3	N	N	N	Y	1	L2D victim buffer is full
L2D_MISSES	0xcb	Y	Y	Y	Y	1	An L2D miss has been issued to the L3, does not include secondary misses
L2D_INSERT_HITS	0xb1	Y	Y	Y	Y	4	Count Number of Times an Inserting Data Request Hit in the L2D.
L2D_INSERT_MISSES	0xb0	Y	Y	Y	Y	4	Count Number of Times an Inserting Data Request Missed in the L2D.

Table 4-20. Derived Monitors for L2 Data Cache Events

Symbol Name	Description	Equation
L2D_READS	L2 Data Read Requests	L2D_REFERENCES.READS
L2D_WRITES	L2 Data Write Requests	L2D_REFERENCES.WRITES
L2D_MISS_RATIO	Percentage of L2D Misses	L2D_INSERT_MISSES/L2D_REFERENCES
L2D_HIT_RATIO	Percentage of L2D Hits	L2D_INSERT_HITS/L2D_REFERENCES
L2D_RECIRC_ATTEMPTS	Number of times the L2 issue logic attempted to issue a recirculate	L2D_ISSUED_RECIRC_OZQ_ACC + L2D_OZQ_CANCEL_S0.RECIRC

4.8.4.1 L2 Data Cache Events (set 0)

Table 4-21. Performance Monitors for L2 Data Cache Set 0

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2D_OZQ_FULL	0xe1 0xe3	N	N	N	1	L2D OZQ Full-ActiveApprox
L2D_OZQ_CANCELS0	0xe0	Y	Y	Y	4	L2D OZQ cancels-TrueThrd
L2D_OZQ_CANCELS1	0xe2	Y	Y	Y	4	L2D OZQ cancels-TrueThrd

L2D_OZQ_FULL is not .all capable.

4.8.4.2 L2 Data Cache Events (set 1)

Table 4-22. Performance Monitors for L2 Data Cache Set 1

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2D_BYPASS	0xe4	Y	Y	Y	4	Count L2 Hit bypasses-TrueThread
L2D_OZQ_RELEASE	0xe5	N	N	N	1	Effective Release is valid in Ozq-ActiveApprox

The L2D_BYPASS count on Itanium 2 processors was too speculative to be useful. It has been fixed and we now count how many bypasses occurred in a given cycle, rather than signalling a 1 for 1-4 bypasses. The 5 and 7 cycle umasks of L2D_BYPASS and the L2D_OZQ_RELEASE counts are not .all capable.

4.8.4.3 L2 Data Cache Events (set 2)

Table 4-23. Performance Monitors for L2 Data Cache Set 2

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2D_REFERENCES	0xe6	Y	Y	Y	4	Inserts of Data Accesses into Ozq-ActiveTrue

4.8.4.4 L2 Data Cache Events (set 3)

Table 4-24. Performance Monitors for L2 Data Cache Set 3

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2D_L3_ACCESS_CANCEL	0xe8	Y	Y	Y	1	L2D request to L3 was cancelled-TrueThrd
L2D_OZDB_FULL	0xe9	N	N	N	1	L2D OZ data buffer is full-AcitveApprox

L2D_L3_ACCESS_CANCEL events are not .all capable.

4.8.4.5 L2 Data Cache Events (set 4)

Table 4-25. Performance Monitors for L2 Data Cache Set 4

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2D_FORCE_RECIRC	0xea	Y	Y	Y	4	Forced recirculates - ActiveTrue or ActiveApprox
L2D_ISSUED_RECIRC_OZQ_ACC	0xeb	Y	Y	Y	1	Ozq Issued Recirculate - TrueThrd

Some umasks of L2D_FORCE_RECIRC are not .all capable.

4.8.4.6 L2 Data Cache Events (set 5)

Table 4-26. Performance Monitors for L2 Data Cache Set 5

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2D_BAD_LINES_SELECTED	0xec	Y	Y	Y	4	Valid line replaced when invalid line is available
L2D_STORE_HIT_SHARED	0xed	Y	Y	Y	2	Store hit a shared line

4.8.4.7 L2 Data Cache Events (set 6)

Table 4-27. Performance Monitors for L2 Data Cache Set 6

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2D_OZQ_ACQUIRE	0xef	N	N	N	1	Valid acquire operation in Ozq-TrueThrd

4.8.4.8 L2 Data Cache Events (set 7)

Table 4-28. Performance Monitors for L2 Data Cache Set 7

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2D_OPS_ISSUED	0xf0	Y	Y	Y	4	Different operations issued by L2D-TrueThrd
L2D_FILLB_FULL	0xf1	N	N	N	1	L2D Fill buffer is full-ActiveApprox

L2D_OPS_ISSUED and L2D_FILLB_FULL are not .all capable.

4.8.4.9 L2 Data Cache Events (set 8)

Table 4-29. Performance Monitors for L2 Data Cache Set 8

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2D_FILL_MESI_STATE	0xf2	Y	Y	Y	1	Fill to L2D is of a particular MESI value. TrueThrd
L2D_VICTIMB_FULL	0xf3	N	N	N	1	L2D victim buffer is full-ActiveApprox

4.8.4.10 L2 Data Cache Events (Not Set Restricted)

These events are sent to the PMU block directly and thus are not set restricted.

Table 4-30. Performance Monitors for L2D Cache - Not Set Restricted

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2D_MISSES	0xcb	Y	Y	Y	1	An L2D miss has been issued to the L3, does not include secondary misses.
L2D_INSERT_MISSES	0xb0	Y	Y	Y	4	An inserting Ozq op was a miss on its first lookup.
L2D_INSERT_HITS	0xb1	Y	Y	Y	4	An inserting Ozq op was a hit on its first lookup.

4.8.5 L3 Cache Events

Table 4-31 summarizes the directly-measured L3 cache events. An extensive list of derived events is provided in Table 4-32.

Table 4-31. Performance Monitors for L3 Unified Cache Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L3_LINES_REPLACED	0xdf	N	N	N	1	L3 Cache Lines Replaced **MESI filtering capability**
L3_MISSES	0xdc	Y	Y	Y	1	L3 Misses
L3_READS	0xdd	Y	Y	Y	1	L3 Reads **MESI filtering capability**
L3_REFERENCES	0xdb	Y	Y	Y	1	L3 References
L3_WRITES	0xde	Y	Y	Y	1	L3 Writes **MESI filtering capability**
L3_INSERTS	0xda				1	L3 Cache lines inserted (allocated) **MESI filtering capability**

Table 4-32. Derived Monitors for L3 Unified Cache Events

Symbol Name	Description	Equation
L3_DATA_HITS	L3 Data Read Hits	L3_READS.DATA_READ.HIT
L3_DATA_MISS_RATIO	L3 Data Miss Ratio	(L3_READS.DATA_READ.MISS + L3_WRITES.DATA_WRITE.MISS) / (L3_READS.DATA_READ.ALL + L3_WRITES.DATA_WRITE.ALL)
L3_DATA_READ_MISSES	L3 Data Read Misses	L3_READS.DATA_READ.MISS
L3_DATA_READ_RATIO	Ratio of L3 References that are Data Read References	L3_READS.DATA_READ.ALL / L3_REFERENCES
L3_DATA_READ_REFERENCES	L3 Data Read References	L3_READS.DATA_READ.ALL
L3_INST_HITS	L3 Instruction Hits	L3_READS.INST_FETCH.HIT
L3_INST_MISSES	L3 Instruction Misses	L3_READS.INST_FETCH.MISS
L3_INST_MISS_RATIO		L3_READS.INST_FETCH.MISS / L3_READS.INST_FETCH.ALL
L3_INST_RATIO	Ratio of L3 References that are Instruction References	L3_READS.INST_FETCH.ALL / L3_REFERENCES
L3_INST_REFERENCES	L3 Instruction References	L3_READS.INST_FETCH.ALL
L3_MISS_RATIO	Percentage Of L3 Misses	L3_MISSES/L3_REFERENCES
L3_READ_HITS	L3 Read Hits	L3_READS.READS.HIT
L3_READ_MISSES	L3 Read Misses	L3_READS.READS.MISS
L3_READ_REFERENCES	L3 Read References	L3_READS.READS.ALL
L3_STORE_HITS	L3 Store Hits	L3_WRITES.DATA_WRITE.HIT
L3_STORE_MISSES	L3 Store Misses	L3_WRITES.DATA_WRITE.MISS
L3_STORE_REFERENCES	L3 Store References	L3_WRITES.DATA_WRITE.ALL
L2_WB_HITS	L2D Writeback Hits	L3_WRITES.L2_WB.HIT
L2_WB_MISSES	L2D Writeback Misses	L3_WRITES.L2_WB.MISS
L2_WB_REFERENCES	L2D Writeback References	L3_WRITES.L2_WB.ALL
L3_WRITE_HITS	L3 Write Hits	L3_WRITES.ALL.HIT
L3_WRITE_MISSES	L3 Write Misses	L3_WRITES.ALL.MISS
L3_WRITE_REFERENCES	L3 Write References	L3_WRITES.ALL.ALL

4.9 System Events

The debug register match events count how often the address of any instruction or data breakpoint register (IBR or DBR) matches the current retired instruction pointer (CODE_DEBUG_REGISTER_MATCHES) or the current data memory address (DATA_DEBUG_REGISTER_MATCHES). CPU_CPL_CHANGES counts the number of privilege level transitions due to interruptions, system calls (epc), returns (demoting branch), and `rfi` instructions.

Table 4-33. Performance Monitors for System Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
CPU_CPL_CHANGES	0x13	N	N	N	1	Privilege Level Changes
DATA_DEBUG_REGISTER_FAULT	0x52	N	N	N	1	Fault due to data debug reg. Match to load/store instruction
DATA_DEBUG_REGISTER_MATCHES	0xc6	Y	Y	Y	1	Data debug register matches data address of memory reference
SERIALIZATION_EVENTS	0x53	N	N	N	1	Number of srlz.I instructions
CYCLES_HALTED	0x18	N	N	N	1	Number of core cycles the thread is in low-power halted state. **NOTE: only PMC/PMD10 pair is capable of counting this event**

Table 4-34. Derived Monitors for System Events

Symbol Name	Description	Equation
CODE_DEBUG_REGISTER_MATCHES	Code Debug Register Matches	IA64_TAGGED_INST_RETIRED

4.10 TLB Events

The Montecito processor instruction and data TLBs and the virtual hash page table walker are monitored by the events described in Table 4-35.

L1ITLB_REFERENCES and L1DTLB_REFERENCES are derived from the respective instruction/data cache access events. Note that ITLB_REFERENCES does not include prefetch requests made to the L1I cache (L1I_PREFETCH_READS). This is because prefetches are cancelled when they miss in the ITLB and thus do not trigger VHPT walks or software TLB miss handling. ITLB_MISSES_FETCH and L2DTLB_MISSES count TLB misses. L1ITLB_INSERTS_HPW and DTLB_INSERTS_HPW count the number of instruction/data TLB inserts performed by the virtual hash page table walker.

Table 4-35. Performance Monitors for TLB Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
DTLB_INSERTS_HPW	0xc9	Y	Y	Y	4	Hardware Page Walker inserts to DTLB
HPW_DATA_REFERENCES	0x2d	Y	Y	Y	4	Data memory references to VHPT
L2DTLB_MISSES	0xc1	Y	Y	Y	4	L2DTLB Misses
L1ITLB_INSERTS_HPW	0x48	Y	N	N	1	L1ITLB Hardware Page Walker Inserts
ITLB_MISSES_FETCH	0x47	Y	N	N	1	ITLB Misses Demand Fetch
L1DTLB_TRANSFER	0xc0	Y	Y	Y	1	L1DTLB misses that hit in the L2DTLB for accesses counted in L1D_READS

Table 4-36. Derived Monitors for TLB Events

Symbol Name	Description	Equation
L1DTLB_EAR_EVENTS	Counts the number of L1DTLB events captured by the EAR	DATA_EAR_EVENTS
L2DTLB_MISS_RATIO	L2DTLB miss ratio	L2DTLB_MISSES / DATA_REFERENCES_SET0 or L2DTLB_MISSES / DATA_REFERENCES_SET1
L1DTLB_REFERENCES	L1DTLB References	DATA_REFERENCES_SET0 or DATA_REFERENCES_SET1
L1ITLB_EAR_EVENTS	Provides information on the number of L1ITLB events captured by the EAR. This is a subset of L1I_EAR_EVENTS	L1I_EAR_EVENTS
L1ITLB_MISS_RATIO	L1ITLB miss ratio	ITLB_MISSES_FETCH.L1ITLB / L1I_READS
L1ITLB_REFERENCES	L1ITLB References	L1I_READS
L1DTLB_FOR_L1D_MISS_RATIO	Miss Ratio of L1DTLB servicing the L1D	L1DTLB_TRANSFER / L1D_READS_SET0 or L1DTLB_TRANSFER / L1D_READS_SET1

The Montecito processor has 2 data TLBs called L1DTLB and L2DTLB (also referred to as DTLB). These TLBs are in parallel and the L2DTLB is the larger and slower of the two.The possible actions for the combination of hits and misses in these TLBs are outlined below:

- L1DTLB_hit=0, L2DTLB_hit=0: If enabled, HPW kicks in and inserts a translation into one or both TLBs.
- L1DTLB_hit=0, L2DTLB_hit=1: If floating-point, no action is taken; else a transfer is made from L2DTLB to L1DTLB.
- L1DTLB_hit=1, L2DTLB_hit=0: If enabled, HPW kicks in and inserts a translation into one or both TLBs.
- L1DTLB_hit=1, L2DTLB_hit=1: No action is taken.

When a memory operation goes down the memory pipeline, DATA_REFERENCES will count it. If the translation does not exist in the L2DTLB, then L2DTLB_MISSES will count it. If the HPW is enabled, then HPW_DATA_REFERENCES will count it. If the HPW finds the data in VHPT, it will insert it in the L1DTLB and L2DTLB (as needed). If the translation exists in the L2DTLB, the only case that some work is done is when translation does not exist in the L1DTLB. If the operation is serviced by the L1D (see L1D_READS description), L1DTLB_TRANSFER will count it. For the purpose of calculating the TLB miss ratios, VHPT memory references have been excluded from the DATA_REFERENCES event and provided VHPT_REFERENCES for the situations where one might want to add them in.

Due to the TLB hardware design, there are some corner cases, where some of these events will show activity even though the instruction causing the activity never reaches retirement (they are marked so). Since the processor is stalled even for these corner cases, they are included in the counts and as long as all events that are used for calculating a metric are consistent with respect to this issue, fairly accurate numbers are expected.

4.11 System Bus Events

Table 4-40 lists the system bus transaction monitors. Many of the listed bus events take a umask that qualifies the event by initiator. For all bus events, when “per cycles” is mentioned, SI clock cycles (bus clock multiplied by bus ratio) are inferred rather than bus clock cycles unless otherwise specified. Numerous derived events have been included in Table 4-41.

Table 4-37. Performance Monitors for System Bus Events (Sheet 1 of 3)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
BUS_ALL	0x87	N	N	N	1	Bus Transactions
ER_BRQ_LIVE_REQ_HI	0xb8	N	N	N	2	BRQ Live Requests (two most-significant-bit of the 5-bit outstanding BRQ request count)
ER_BRQ_LIVE_REQ_LO	0xb9	N	N	N	7	BRQ Live Requests (three least-significant-bit of the 5-bit outstanding BRQ request count)
ER_BRQ_REQ_INSERTED	0xba	N	N	N	1	BRQ Requests Inserted
ER_BKSNP_ME_ACCEPTED	0xbb	N	N	N	1	BacksnoopMe Requests accepted into the BRQ from the L2D (used by the L2D to get itself out of potential forward progress situations)
ER_REJECT_ALL_L1_REQ	0xbc	N	N	N	1	Number of cycles in which the BRQ was rejecting all L1I/L1D requests (for the “Big Hammer” forward progress logic)
ER_REJECT_ALL_L1D_REQ	0xbd	N	N	N	1	Number of cycles in which the BRQ was rejecting all L1D requests (for L1D/L1I forward progress)
ER_REJECT_ALL_L1I_REQ	0xbe	N	N	N	1	Number of cycles in which the BRQ was rejecting all L1I requests (for L1D/L1I forward progress)
BUS_DATA_CYCLE	0x88	N	N	N	1	Valid data cycle on the Bus
BUS_HITM	0x84	N	N	N	1	Bus Hit Modified Line Transactions
BUS_IO	0x90	N	N	N	1	IA-32 Compatible IO Bus Transactions
SI_IOQ_LIVE_REQ_HI	0x98	N	N	N	1	In-order Bus Queue Requests (one most-significant-bit of the 4-bit outstanding IOQ request count)
SI_IOQ_LIVE_REQ_LO	0x97	N	N	N	7	In-order Bus Queue Requests (three least-significant-bit of the 4-bit outstanding IOQ request count)
BUS_B2B_DATA_CYCLES	0x93	N	N	N	1	Back-to-back bursts of data
SI_CYCLES	0x8e	N	N	N	1	Counts SI cycles
BUS_MEMORY	0x8a	N	N	N	1	Bus Memory Transactions
BUS_MEM_READ	0x8b	N	N	N	1	Full Cache line D/I memory RD, RD invalidate, and BRIL
ER_MEM_READ_OUT_HI	0xb4	N	N	N	2	Outstanding memory RD transactions (upper two bits)

Table 4-37. Performance Monitors for System Bus Events (Sheet 2 of 3)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
ER_MEM_READ_OUT_LO	0xb5	N	N	N	7	Outstanding memory RD transactions (lower three bits)
BUS_RD_DATA	0x8c	N	N	N	1	Bus Read Data Transactions
BUS_RD_HIT	0x80	N	N	N	1	Bus Read Hit Clean Non-local Cache Transactions
BUS_RD_HITM	0x81	N	N	N	1	Bus Read Hit Modified Non-local Cache Transactions
BUS_RD_INVAL_BST_HITM	0x83	N	N	N	1	Bus BRIL Burst Transaction Results in HITM
BUS_RD_INVAL_HITM	0x82	N	N	N	1	Bus BIL Transaction Results in HITM
BUS_RD_IO	0x91	N	N	N	1	IA-32 Compatible IO Read Transactions
BUS_RD_PRTL	0x8d	N	N	N	1	Bus Read Partial Transactions
ER_SNOOPQ_REQ_HI	0xb6	N	N	N	1	ER Snoop Queue Requests (most significant bit of 4-bit count)
ER_SNOOPQ_REQ_LO	0xb7	N	N	N	7	ER Snoop Queue Requests (three least-significant-bits or 4-bit count)
BUS_SNOOP_STALL_CYCLES	0x8f	N	N	N	1	Bus Snoop Stall Cycles (from any agent)
BUS_WR_WB	0x92	N	N	N	1	Bus Write Back Transactions
MEM_READ_CURRENT	0x89	N	N	N	1	Current Mem Read Transactions On Bus
SI_RQ_INSERTS	0x9e	N	N	N	2	SI request queue inserts
SI_RQ_LIVE_REQ_HI	0xa0	N	N	N	1	SI request queue live requests (most-significant bit)
SI_RQ_LIVE_REQ_LO	0x9f	N	N	N	7	SI request queue live requests (least-significant three bits)
SI_WRITEQ_INSERTS	0xa1	N	N	N	2	SI write queue inserts
SI_WRITEQ_LIVE_REQ_HI	0xa3	N	N	N	1	SI write queue live requests (most-significant bit)
SI_WRITEQ_LIVE_REQ_LO	0xa2	N	N	N	7	SI write queue live requests (least-significant three bits)
SI_WAQ_COLLISIONS	0xa4	N	N	N	1	SI write address queue collisions (incoming FSB snoop collides with an entry in WAQ)
SI_CCQ_INSERTS	0xa5	N	N	N	2	SI clean castout queue inserts
SI_CCQ_LIVE_REQ_HI	0xa7	N	N	N	1	SI clean castout queue live requests (most-significant bit)
SI_CCQ_LIVE_REQ_LO	0xa6	N	N	N	7	SI clean castout queue live requests (least-significant three bits)
SI_CCQ_COLLISIONS	0xa8	N	N	N	1	SI clean castout queue collisions (incoming FSB snoop collides with an entry in CCQ)
SI_IOQ_COLLISIONS	0xaa	N	N	N	1	SI inorder queue collisions (outgoing transaction collides with an entry in IOQ)

Table 4-37. Performance Monitors for System Bus Events (Sheet 3 of 3)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
SI_SCB_INSERTS	0xab	N	N	N	1	SI snoop coalescing buffer inserts
SI_SCB_LIVE_REQ_HI	0xad	N	N	N	1	SI snoop coalescing buffer live requests (most-significant bit)
SI_SCB_LIVE_REQ_LO	0xac	N	N	N	7	SI snoop coalescing buffer live requests (least-significant three bits)
SI_SCB_SIGNOFFS	0xae	N	N	N	1	SI snoop coalescing buffer coherency signoffs
SI_WDQ_ECC_ERRORS	0xaf	N	N	N	1	SI write data queue ECC errors

Table 4-38. Derived Monitors for System Bus Events (Sheet 1 of 2)

Symbol Name	Description	Equation
BIL_HITM_LINE_RATIO	BIL Hit to Modified Line Ratio	BUS_RD_INVAL_HITM / BUS_MEMORY or BUS_RD_INVAL_HITM / BUS_RD_INVAL
BIL_RATIO	BIL Ratio	BUS_RD_INVAL / BUS_MEMORY
BRIL_HITM_LINE_RATIO	BRIL Hit to Modified Line Ratio	BUS_RD_INVAL_BST_HITM / BUS_MEMORY or BUS_RD_INVAL_BST_HITM / BUS_RD_INVAL
BUS_ADDR_BPRI	Bus transactions used by IO agent.	BUS_MEMORY.*.IO
BUS_BRQ_LIVE_REQ	BRQ Live Requests	ER_BRQ_LIVE_REQ_HI * 8 + ER_BRQ_LIVE_REQ_LO
BUS_BURST	Full cache line memory transactions (BRL, BRIL, BWL)	BUS_MEMORY.EQ_128BYTE.*
BUS_HITM_RATIO	Bus Modified Line Hit Ratio	BUS_HITM / BUS_MEMORY or BUS_HITM / BUS_BURST
BUS_HITS_RATIO	Bus Read Hit to Shared Line Ratio	BUS_RD_HIT / BUS_RD_ALL or BUS_RD_HIT / BUS_MEMORY
BUS_IOQ_LIVE_REQ	Inorder Bus Queue Requests	SI_IOQ_LIVE_REQ_HI * 8+ SI_IOQ_LIVE_REQ_LO
BUS_IO_CYCLE_RATIO	Bus I/O Cycle Ratio	BUS_IO / BUS_ALL
BUS_IO_RD_RATIO	Bus I/O Read Ratio	BUS_RD_IO / BUS_IO
BUS_MEM_READ_OUTSTANDING	Number of outstanding memory RD transactions	ER_MEM_READ_OUT_HI * 8 + ER_MEM_READ_OUT_LO
BUS_PARTIAL	Less than cache line memory transactions (BRP, BWP)	BUS_MEMORY.LT_128BYTE.*
BUS_PARTIAL_RATIO	Bus Partial Access Ratio	BUS_MEMORY.LT_128BYTE / BUS_MEMORY.ALL
BUS_RD_ALL	Full cache line memory read transactions (BRL)	BUS_MEM_READ.BRL.*
BUS_RD_DATA_RATIO	Cacheable Data Fetch Bus Transaction Ratio	BUS_RD_DATA / BUS_ALL or BUS_RD_DATA / BUS_MEMORY

Table 4-38. Derived Monitors for System Bus Events (Sheet 2 of 2)

Symbol Name	Description	Equation
BUS_RD_HITM_RATIO	Bus Read Hit to Modified Line Ratio	BUS_RD_HITM / BUS_RD_ALL or BUS_RD_HITM / BUS_MEMORY
BUS_RD_INSTRUCTIONS	Full cache line instruction memory read transactions (BRP)	BUS_RD_ALL - BUS_RD_DATA
BUS_RD_INVAL	0 byte memory read-invalidate transactions (BIL)	BUS_MEM_READ.BIL.*
BUS_RD_INVAL_BST	Full cache line read-invalidate transactions (BRIL)	BUS_MEM_READ.BRIL.*
BUS_RD_INVAL_BST_MEMORY	Bus Read Invalid Line in Burst transactions (BRIL) satisfied by memory	BUS_RD_INVAL_BST - BUS_RD_INVAL_BST_HITM
BUS_RD_INVAL_MEMORY	Bus Read Invalidate Line transactions (BIL) satisfied from memory	BUS_RD_INVAL - BUS_RD_INVAL_HITM
BUS_RD_INVAL_ALL_HITM	Bus Read Invalidate Line transactions (BRIL and BIL) resulting in HITMs	BUS_RD_INVAL_BST_HITM + BUS_RD_INVAL_HITM
BUS_RD_PRTL_RATIO	Bus Read Partial Access Ratio	BUS_RD_PRTL / BUS_MEMORY
BUS_WB_RATIO	Writeback Ratio	BUS_WR_WB / BUS_MEMORY or BUS_WR_WB / BUS_BURST
CACHEABLE_READ_RATIO	Cacheable Read Ratio	(BUS_RD_ALL + BUS_MEM_READ.BRIL) / BUS_MEMORY

4.11.1 System Bus Conventions

Table 4-39 defines the conventions that will be used when describing the Montecito processor system bus transaction monitors in this section, as well as the individual monitor descriptions in Section 4.15.

Other transactions besides those listed in Table 4-42 include Deferred Reply, Special Transactions, Interrupt, Interrupt Acknowledge, and Purge TC. Note that the monitors will count if any transaction gets a retry response from the priority agent.

To support the analysis of snoop traffic in a multiprocessor system, the Montecito processor provides local processor and remote response monitors. The local processor snoop events (SI_SCB_INSERTS and SI_SCB_SIGNOFFS) monitor inbound snoop traffic. The remote response events (BUS_RD_HIT, BUS_RD_HITM, BUS_RD_INVAL_HITM and BUS_RD_INVAL_BST_HITM) monitor the snoop responses of other processors to bus transactions that the monitoring processor originated. Table 4-40 summarizes the remote snoop events by bus transaction.

4.11.2 Extracting Memory Latency from Montecito Performance Counters

On the Itanium 2 processors, several events were provided to approximate memory latency as seen by the processor using the following equation:

((BUS_MEM_READ_OUT_HI * 8) + BUS_MEM_READ_OUT_LO) /
(BUS_MEM_READ.BRL.SELF + BUS_MEM_READ.BRIL.SELF)

The BUS_MEM_READ_OUT starts counting one bus clock after a request is issued on the system interface (ADS) and stops incrementing when the request completes its first data transfer or is retried. In each core cycle after counting is initiated, the number of live requests in that cycle are added to the count. This count may as high as 15. For ease of implementation, the count is split into two parts: BUS_MEM_READ_OUT_LO sums up the low order 3 bits of the number of live requests, while BUS_MEM_READ_OUT_HI sums up the high order bit.

In the above formula, the numerator provides the number of live requests and the denominator provides the number of requests that are counted. When the live count is divided by the number of transactions issued, you get an average lifetime of a transaction issued on the system interface (a novel application of Little's Law).

The Montecito processor has similar counters: ER_MEM_READ_OUT.{HI,LO}. Using these events to derive Montecito memory latency will give results that are higher than the true memory latency seen in Montecito. The main reason for this is the fact that the start and stop point of the counters are not equivalent between the two processors. Specifically, in Montecito, ER_MEM_READ_OUT.{HI,LO}events start counting the core clock after a request is sent to the arbiter. The Montecito ER_MEM_READ_OUT_{HI,LO}events stop counting when the request receives its first data transfer within the external request logic (after the arbiter). Thus, these events include the entire time requests spend in the arbiter (pre and post request).

The requests may remain in the arbiter for a long or short time depending on system interface behaviors. Arbiter queue events SI_RQ_LIVE_REQ.{HI,LO} may be used to reduce the effects of arbiter latency on the calculations. Unfortunately, these events are not sufficient to successfully enable a completely equivalent measurement for Itanium 2 processors. The arbiter time back from FSB to core is fixed for a specific arbiter to system interface ratio. These arbiter events may occur in a different time domain from core events

The new memory latency approximation formula for Montecito, with corrective events included, is below:

(ER_MEM_READ_OUT_HI * 8 + ER_MEM_READ_OUT_LO) – (SI_RQ_LIVE_REQ_HI * 8 + SI_RQ_LIVE_REQ_LO) / (BUS_MEM_READ)

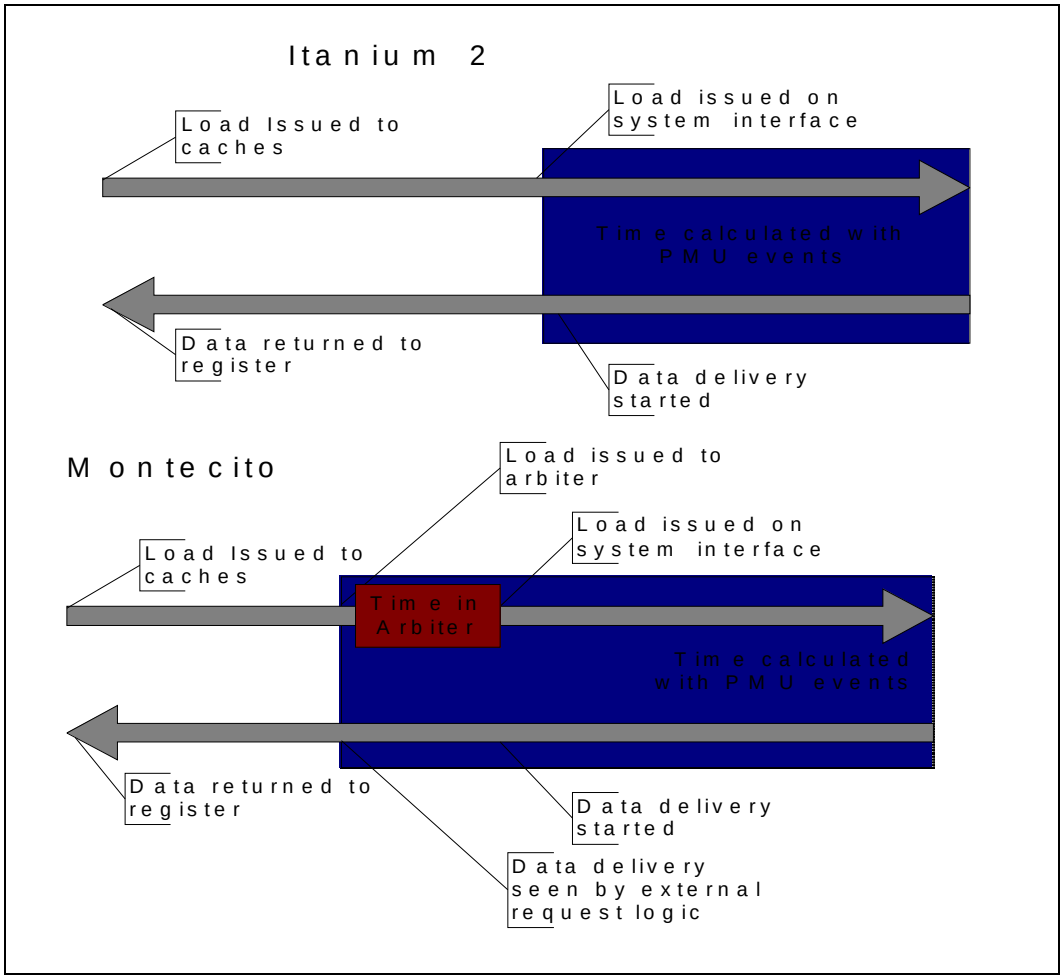
Note that the Data EAR may be used to compare data cache load miss latency between Madison and Montecito. However, an access' memory latency, as measured by the Data EAR or other cycle counters will be inherently greater on Montecito compared to previous Itanium 2 processors due to the latency the arbiter adds to both the outbound request and inbound data transfer. Also, the Data EAR encompasses the entire latency through the processor's memory hierarchy and queues without details into time spent in any specific queue.

Even with this improved formula, the estimated memory latency for Montecito will appear greater than previous Itanium 2 processors. We have not observed any design point that suggests that the system interface component of memory accesses are excessive on Montecito.

We have observed that snoop stalls and write queue pressure lead to additional memory latency on Montecito compared to previous Itanium 2 processors, but these are phenomena that impact the pre-system or post-system interface aspect of a memory latency and are very workload dependant in their impact. Specifically, the write queues need to be sufficiently filled to cause back pressure on the victimizing read requests such that a new read request cannot issue to the system interface because it cannot identify a victim in the L3 cache to ensure its proper allocation. This sever pressure has only been seen with steady streams of every read requests resulting in a dirty L3 victim. Additional snoop stalls should only add latency to transactions that receive a HITM snoop response (cache to cache transfers) because non-HITM responses are satisfied by the memory and memory access should be initiated as a consequence of the initial transaction rather than its snoop response.

The figure below (Figure 4-2) shows the latency is determined using the above calculations on Itanium 2 and Montecito processors. The red portion of the Montecito diagram shows latency accounted for by the correction found in the Montecito calculation.

Figure 4-2. Extracting Memory Latency from PMUs



4.12 RSE Events

Register Stack Engine events are presented in Table 4-39. The number of current/dirty registers are split among three monitors since there are 96 physical registers in the Montecito processor.

Table 4-39. Performance Monitors for RSE Events (Sheet 1 of 2)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
RSE_CURRENT_REGS_2_TO_0	0x2b	N	N	N	7	Current RSE registers
RSE_CURRENT_REGS_5_TO_3	0x2a	N	N	N	7	Current RSE registers
RSE_CURRENT_REGS_6	0x26	N	N	N	1	Current RSE registers
RSE_DIRTY_REGS_2_TO_0	0x29	N	N	N	7	Dirty RSE registers

Table 4-39. Performance Monitors for RSE Events (Sheet 2 of 2)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
RSE_DIRTY_REGS_5_TO_3	0x28	N	N	N	7	Dirty RSE registers
RSE_DIRTY_REGS_6	0x24	N	N	N	1	Dirty RSE registers
RSE_EVENT_RETIRED	0x32	N	N	N	1	Retired RSE operations
RSE_REFERENCES_RETIRED	0x20	Y	Y	Y	2	RSE Accesses

Table 4-40. Derived Monitors for RSE Events

Symbol Name	Description	Equation
RSE_CURRENT_REGS	Current RSE registers before an RSE_EVENT_RETIRED occurred	RSE_CURRENT_REGS_6 * 64 + RSE_CURRENT_REGS_5_TO_3 * 8 + RSE_CURRENT_REGS_2_TO_0
RSE_DIRTY_REGS	Dirty RSE registers before an RSE_EVENT_RETIRED occurred	RSE_DIRTY_REGS_6 * 64 + RSE_DIRTY_REGS_5_TO_3 * 8 + RSE_DIRTY_REGS_2_TO_0
RSE_LOAD_LATENCY_PENALTY	Counts the number of cycles we have stalled due to retired RSE loads. (Every time RSE.BOF reaches RSE.storereg and RSE has not issued all of the loads necessary for the fill.)	BE_RSE_BUBBLE.UNDERFLOW
RSE_AVG_LOAD_LATENCY	Average latency for RSE loads	RSE_LOAD_LATENCY_PENALTY / RSE_REFERENCES_RETIRED.LOAD
RSE_AVG_CURRENT_REGS	Average number of current registers	RSE_CURRENT_REGS / RSE_EVENT_RETIRED
RSE_AVG_DIRTY_REGS	Average number of dirty registers	RSE_DIRTY_REGS / RSE_EVENT_RETIRED
RSE_AVG_INVALID_REGS	Average number of invalid registers. Assumes number of clean registers is always 0.	96 - (RSE_DIRTY_REGS + RSE_CURRENT_REGS) / RSE_EVENT_RETIRED

4.13 Hyper-Threading Events

Table 4-41 summarizes the events available on the Montecito processor to measure thread switch activity. To determine the raw number of thread switch events that occurs during a given monitoring session, a user should capture THREAD_SWITCH_EVENTS.ALL.

Table 4-41. Performance Monitors for Multi-thread Events (Sheet 1 of 2)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
THREAD_SWITCH_EVENTS	0x0c	N	N	N	1	Counts number of times the thread is switched out due to various causes.
THREAD_SWITCH_GATED	0x0d	N	N	N	1	Number of times thread switch outs are gated and their causes

Table 4-41. Performance Monitors for Multi-thread Events (Sheet 2 of 2)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
THREAD_SWITCH_CYCLE	0x0e	N	N	N	1	Various overhead cycles spent for thread switches
THREAD_SWITCH_STALL	0x0f	N	N	N	1	Times main pipeline is stalled more than a threshold value set before a thread switch

4.14 Performance Monitors Ordered by Event Code

Table 4-42 presents all of the performance monitors provided in the Montecito processor ordered by their event code.

Table 4-42. All Performance Monitors Ordered by Code (Sheet 1 of 7)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Type	Description
BACK_END_BUBBLE	0x00	N	N	N	1	A	Full pipe bubbles in main pipe
BE_RSE_BUBBLE	0x01	N	N	N	1	A	Full pipe bubbles in main pipe due to RSE stalls
BE_EXE_BUBBLE	0x02	N	N	N	1	A	Full pipe bubbles in main pipe due to Execution unit stalls
FP_TRUE_SIRSTALL	0x03	Y	N	N	1	A	SIR stall asserted and leads to a trap
BE_FLUSH_BUBBLE	0x04	N	N	N	1	A	Full pipe bubbles in main pipe due to flushes
FP_FALSE_SIRSTALL	0x05	Y	N	N	1	A	SIR stall without a trap
FP_FAILED_FCHKF	0x06	Y	N	N	1	A	Failed fchkf
IA64_INST_RETIRED	0x08	Y	N	Y	6	A	Retired Itanium® Instructions
IA64_TAGGED_INST_RETIRED	0x08	Y	N	Y	6	A	Retired Tagged Instructions
FP_OPS_RETIRED	0x09	Y	N	N	6	A	Retired FP operations
FP_FLUSH_TO_ZERO	0x0b	Y	N	N	2	A	FP Result Flushed to Zero
THREAD_SWITCH_EVENTS	0x0c	N	N	N	1	A	Thread switch events and cause
THREAD_SWITCH_GATED	0x0d	N	N	N	1	A	TS gated and their sources
THREAD_SWITCH_CYCLE	0x0e	N	N	N	1	A	Various TS related periods
THREAD_SWITCH_STALLS	0x0f	N	N	N	1	A	Pipe line stalls due to TS
BRANCH_EVENT	0x11	Y	N	Y	1	A	Branch Event Captured
CPU_OP_CYCLES	0x12	Y	N	Y	1	C	CPU Operating Cycles
CPU_CPL_CHANGES	0x13	N	N	N	1	A	Privilege Level Changes
CPU_OP_CYCLES_HALTED	0x18	N	N	N	7	C	CPU Operating Cycles Halted
RSE_REFERENCES_RETIRED	0x20	Y	Y	Y	2	A	RSE Accesses
RSE_DIRTY_REGS_6	0x24	N	N	N	1	A	Dirty RSE registers

Table 4-42. All Performance Monitors Ordered by Code (Sheet 2 of 7)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Type	Description
RSE_CURRENT_REGS_6	0x26	N	N	N	1	A	Current RSE registers
RSE_DIRTY_REGS_5_TO_3	0x28	N	N	N	7	A	Dirty RSE registers
RSE_DIRTY_REGS_2_TO_0	0x29	N	N	N	7	A	Dirty RSE registers
RSE_CURRENT_REGS_5_TO_3	0x2a	N	N	N	7	A	Current RSE registers
RSE_CURRENT_REGS_2_TO_0	0x2b	N	N	N	7	A	Current RSE registers
HPW_DATA_REFERENCES	0x2d	Y	Y	Y	4	A	Data memory references to VHPT
RSE_EVENT_RETIRED	0x32	N	N	N	1	A	Retired RSE operations
L1I_READS	0x40	Y	N	N	1	A	L1 Instruction Cache Reads
L1I_FILLS	0x41	Y	N	N	1	F	L1 Instruction Cache Fills
L2I_DEMAND_READS	0x42	Y	N	N	1	A	L1 Instruction Cache and ISB Misses
L1I_EAR_EVENTS	0x43	Y	N	N	1	F	Instruction EAR Events
L1I_PREFETCHES	0x44	Y	N	N	1	A	L1 Instruction Prefetch Requests
L2I_PREFETCHES	0x45	Y	N	N	1	A	L2 Instruction Prefetch Requests
ISB_BUNPAIRS_IN	0x46	Y	N	N	1	F	Bundle pairs written from L2 into FE
ITLB_MISSES_FETCH	0x47	Y	N	N	1	A	ITLB Misses Demand Fetch
L1ITLB_INSERTS_HPW	0x48	Y	N	N	1	A	L1ITLB Hardware Page Walker Inserts
DISP_STALLED	0x49	N	N	N	1	A	Number of cycles dispersal stalled
L1I_SNOOP	0x4a	Y	Y	Y	1	C	Snoop requests handled by L1I
L1I_PURGE	0x4b	Y	N	N	1	C	L1ITLB purges handled by L1I
INST_DISPERSED	0x4d	Y	N	N	6	A	Syllables Dispersed from REN to REG stage
SYLL_NOT_DISPERSED	0x4e	Y	N	N	5	A	Syllables not dispersed
SYLL_OVERCOUNT	0x4f	Y	N	N	2	A	Syllables overcounted
NOPS_RETIRED	0x50	Y	N	Y	6	A	Retired NOP Instructions
PREDICATE_SQUASHED_RETIRED	0x51	Y	N	Y	6	A	Instructions Squashed Due to Predicate Off
DATA_DEBUG_REGISTER_FAULT	0x52	N	N	N	1	A	Fault due to data debug reg. Match to load/store instruction
SERIALIZATION_EVENTS	0x53	N	N	N	1	A	Number of srlz.i instructions
BR_PATH_PRED	0x54	Y	N	Y	3	A	FE Branch Path Prediction Detail
INST_FAILED_CHKS_RETIRED	0x55	N	N	N	1	A	Failed Speculative Check Loads
INST_CHKA_LDC_ALAT	0x56	Y	Y	Y	2	A	Advanced Check Loads
INST_FAILED_CHKA_LDC_ALAT	0x57	Y	Y	Y	1	A	Failed Advanced Check Loads
ALAT_CAPACITY_MISS	0x58	Y	Y	Y	2	A	ALAT Entry Replaced
BR_MISPRED_DETAIL	0x5b	Y	N	Y	3	A	FE Branch Mispredict Detail
L1I_STRM_PREFETCHES	0x5f	Y	N	N	1	A	L1 Instruction Cache line prefetch requests

Table 4-42. All Performance Monitors Ordered by Code (Sheet 3 of 7)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Type	Description
L1I_RAB_FULL	0x60	N	N	N	1	C	Is RAB full?
BE_BR_MISPRED_DETAIL	0x61	Y	N	Y	1	A	BE branch misprediction detail
ENCBR_MISPRED_DETAIL	0x63	Y	N	Y	3	A	Number of encoded branches retired
L1I_RAB_ALMOST_FULL	0x64	N	N	N	1	C	Is RAB almost full?
L1I_FETCH_RAB_HIT	0x65	Y	N	N	1	A	Instruction fetch hitting in RAB
L1I_FETCH_ISB_HIT	0x66	Y	N	N	1	A	“Just-in-time” instruction fetch hitting in and being bypassed from ISB
L1I_PREFETCH_STALL	0x67	N	N	N	1	A	Why prefetch pipeline is stalled?
BR_MISPRED_DETAIL2	0x68	Y	N	Y	2	A	FE Branch Mispredict Detail (Unknown path component)
L1I_PVAB_OVERFLOW	0x69	N	N	N	1	A	PVAB overflow
BR_PATH_PRED2	0x6a	Y	N	Y	2	A	FE Branch Path Prediction Detail (Unknown prediction component)
FE_LOST_BW	0x70	N	N	N	2	A	Invalid bundles at the entrance to IB
FE_BUBBLE	0x71	N	N	N	1	A	Bubbles seen by FE
BE_LOST_BW_DUE_TO_FE	0x72	N	N	N	2	A	Invalid bundles if BE not stalled for other reasons
IDEAL_BE_LOST_BW_DUE_TO_FE	0x73	N	N	N	2	A	Invalid bundles at the exit from IB
L2I_READS	0x78	Y	N	Y	1	F	L2I Cachebale Reads
L2I_UC_READS	0x79	Y	N	Y	1	F	L2I uncacheable reads
L2I_VICTIMIZATIONS	0x7a	Y	N	Y	1	F	L2I victimizations
L2I_RECIRCULATES	0x7b	Y	N	Y	1	F	L2I recirculates
L2I_L3_REJECTS	0x7c	Y	N	Y	1	F	L3 rejects
L2I_HIT_CONFLICTS	0x7d	Y	N	Y	1	F	L2I hit conflicts
L2I_SPEC_ABORTS	0x7e	Y	N	Y	1	F	L2I speculative aborts
L2I_SNOOP_HITS	0x7f	Y	N	Y	1	C	L2I snoop hits
BUS_RD_HIT	0x80	N	N	N	1	S	Bus Read Hit Clean Non-local Cache Transactions
BUS_RD_HITM	0x81	N	N	N	1	S	Bus Read Hit Modified Non-local Cache Transactions
BUS_RD_INVAL_HITM	0x82	N	N	N	1	S	Bus BIL Transaction Results in HITM
BUS_RD_INVAL_ALL_HITM	0x83	N	N	N	1	S	Bus BIL or BRIL Transaction Results in HITM
BUS_HITM	0x84	N	N	N	1	S	Bus Hit Modified Line Transactions
BUS_ALL	0x87	N	N	N	1	S	Bus Transactions
BUS_DATA_CYCLE	0x88	N	N	N	1	C	Valid data cycle on the Bus
MEM_READ_CURRENT	0x89	N	N	N	1	C	Current Mem Read Transactions On Bus
BUS_MEMORY	0x8a	N	N	N	1	S	Bus Memory Transactions

Table 4-42. All Performance Monitors Ordered by Code (Sheet 4 of 7)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Type	Description
BUS_MEM_READ	0x8b	N	N	N	1	S	Full Cache line D/I memory RD, RD invalidate, and BRIL
BUS_RD_DATA	0x8c	N	N	N	1	S	Bus Read Data Transactions
BUS_RD_PRTL	0x8d	N	N	N	1	S	Bus Read Partial Transactions
SI_CYCLES	0x8e	N	N	N	1	C	SI clock cycles
BUS_SNOOP_STALL_CYCLES	0x8f	N	N	N	1	F	Bus Snoop Stall Cycles (from any agent)
BUS_IO	0x90	N	N	N	1	S	IA-32 Compatible IO Bus Transactions
BUS_RD_IO	0x91	N	N	N	1	S	IA-32 Compatible IO Read Transactions
BUS_WR_WB	0x92	N	N	N	1	S	Bus Write Back Transactions
BUS_B2B_DATA_CYCLES	0x93	N	N	N	1	C	Back-to-back bursts of data
SI_IOQ_LIVE_REQ_LO	0x97	N	N	N	7	C	Inorder Bus Queue Requests (three least-significant-bits of the 4-bit outstanding IOQ request count)
SI_IOQ_LIVE_REQ_HI	0x98	N	N	N	1	C	Inorder Bus Queue Requests (most-significant-bit of the 4-bit outstanding IOQ request count)
SI_L3T_TRACE_CACHE	0x9d	N	N	N	n/a	F	SI PMU wires are used for trace cache data.
SI_RQ_INSERTS	0x9e	N	N	N	2	S	SI request queue inserts
SI_RQ_LIVE_REQ_LO	0x9f	N	N	N	7	C	SI request queue live requests (least-significant three bits)
SI_RQ_LIVE_REQ_HI	0xa0	N	N	N	1	C	SI request queue live requests (most-significant bit)
SI_WRITEQ_INSERTS	0xa1	N	N	N	2	S	SI write queue inserts
SI_WRITEQ_LIVE_REQ_LO	0xa2	N	N	N	7	C	SI write queue live requests (least-significant three bits)
SI_WRITEQ_LIVE_REQ_HI	0xa3	N	N	N	1	C	SI write queue live requests (most-significant bit)
SI_WAQ_COLLISIONS	0xa4	N	N	N	1	C	SI write address queue collisions (incoming FSB snoop collides with an entry in WAQ)
SI_CCQ_INSERTS	0xa5	N	N	N	2	S	SI clean castout queue inserts
SI_CCQ_LIVE_REQ_LO	0xa6	N	N	N	7	C	SI clean castout queue live requests (least-significant three bits)
SI_CCQ_LIVE_REQ_HI	0xa7	N	N	N	1	C	SI clean castout queue live requests (most-significant bit)
SI_CCQ_COLLISIONS	0xa8	N	N	N	1	C	SI clean castout queue collisions (incoming FSB snoop collides with an entry in CCQ)
SI_IOQ_COLLISIONS	0xaa	N	N	N	1	C	SI inorder queue collisions (outgoing transaction collides with an entry in IOQ)

Table 4-42. All Performance Monitors Ordered by Code (Sheet 5 of 7)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Type	Description
SI_SCB_INSERTS	0xab	N	N	N	1	C	SI snoop coalescing buffer inserts
SI_SCB_LIVE_REQ_LO	0xac	N	N	N	7	C	SI snoop coalescing buffer live requests (least-significant three bits)
SI_SCB_LIVE_REQ_HI	0xad	N	N	N	1	C	SI snoop coalescing buffer live requests (most-significant bit)
SI_SCB_SIGNOFFS	0xae	N	N	N	1	C	SI snoop coalescing buffer coherency signoffs
SI_WDQ_ECC_ERRORS	0xaf	N	N	N	1	C	SI write data queue ECC errors
L2D_INSERT_MISSES	0xb0	N	N	N	4	F	Count Number of Times an Inserting Data Request Missed in the L2D.
L2D_INSERT_HITS	0xb1	N	N	N	4	F	Count Number of Times an Inserting Data Request Hit in the L2D.
ER_MEM_READ_OUT_HI	0xb4	N	N	N	2	F	Outstanding memory RD transactions
ER_MEM_READ_OUT_LO	0xb5	N	N	N	7	F	Outstanding memory RD transactions
ER_SNOOPQ_REQ_HI	0xb6	N	N	N	1	C	ER Snoop Queue Requests (most significant bit of 4-bit count)
ER_SNOOPQ_REQ_LO	0xb7	N	N	N	7	C	ER Snoop Queue Requests (least sifnificant three bits of 4-bit count)
ER_BRQ_LIVE_REQ_HI	0xb8	N	N	N	2	C	BRQ Live Requests (two most-significant-bit of the 5-bit outstanding BRQ request count)
ER_BRQ_LIVE_REQ_LO	0xb9	N	N	N	7	C	BRQ Live Requests (three least-significant-bit of the 5-bit outstanding BRQ request count)
ER_BRQ_REQ_INSERTED	0xba	N	N	N	1	F	BRQ Requests Inserted
ER_BKSNP_ME_ACCEPTED	0xbb	N	N	N	1	C	BacksnoopMe Requests accepted into the BRQ from the L2d (used by the L2d to get itself out of potential forward progress situations)
ER_REJECT_ALL_L1_REQ	0xbc	N	N	N	1	C	Number of cycles in which the BRQ was rejecting all L1i/L1d requests (for the “Big Hammer” forward progress logic)
ER_REJECT_ALL_L1D_REQ	0xbd	N	N	N	1	C	Number of cycles in which the BRQ was rejecting all L1d requests (for L1d/L1i forward progress)
ER_REJECT_ALL_L1I_REQ	0xbe	N	N	N	1	C	Number of cycles in which the BRQ was rejecting all L1i requests (for L1d/L1i forward progress)
L1DTLB_TRANSFER	0xc0	Y	Y	Y	1	A	L1DTLB misses that hit in the L2DTLB for accesses counted in L1D_READS
L2DTLB_MISSES	0xc1	Y	Y	Y	4	A	L2DTLB Misses
L1D_READS_SET0	0xc2	Y	Y	Y	2	A	L1 Data Cache Reads

Table 4-42. All Performance Monitors Ordered by Code (Sheet 6 of 7)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Type	Description
DATA_REFERENCES_SET0	0xc3	Y	Y	Y	4	A	Data memory references issued to memory pipeline
L1D_READS_SET1	0xc4	Y	Y	Y	2	A	L1 Data Cache Reads
DATA_REFERENCES_SET1	0xc5	Y	Y	Y	4	A	Data memory references issued to memory pipeline
DATA_DEBUG_REGISTER_MATCHES	0xc6	Y	Y	Y	1	A	Data debug register matches data address of memory reference
L1D_READ_MISSES	0xc7	Y	Y	Y	2	A	L1 Data Cache Read Misses
DATA_EAR_EVENTS	0xc8	Y	Y	Y	1	F	L1 Data Cache EAR Events
DTLB_INSERTS_HPW	0xc9	Y	Y	Y	4	F	Hardware Page Walker inserts to DTLB
BE_L1D_FPU_BUBBLE	0xca	N	N	N	1	A	Full pipe bubbles in main pipe due to FP or L1 dcache
L2D_MISSES	0xcb	Y	Y	Y	1		An L2D miss has been issued to the L3, does not include secondary misses.
LOADS_RETIRED	0xcd	Y	Y	Y	4	A	Retired Loads
MISALIGNED_LOADS_RETIRED	0xce	Y	Y	Y	4	A	Retired Misaligned Load Instructions
UC_LOADS_RETIRED	0xcf	Y	Y	Y	4	A	Retired Uncacheable Loads
UC_STORES_RETIRED	0xd0	Y	Y	Y	2	A	Retired Uncacheable Stores
STORES_RETIRED	0xd1	Y	Y	Y	2	A	Retired Stores
MISALIGNED_STORES_RETIRED	0xd2	Y	Y	Y	2	A	Retired Misaligned Store Instructions
LOADS_RETIRED_INTG	0xd8	Y	Y	Y	2	A	Integer Loads retired
SPEC_LOADS_NATTED	0xd9	Y	Y	Y	2	A	Speculative loads NAT'd
L3_INSERTS	0xda	Y	Y	Y	1	F	L3 Inserts (allocations)
L3_REFERENCES	0xdb	Y	Y	Y	1	F	L3 References
L3_MISSES	0xdc	Y	Y	Y	1	F	L3 Misses
L3_READS	0xdd	Y	Y	Y	1	F	L3 Reads
L3_WRITES	0xde	Y	Y	Y	1	F	L3 Writes
L3_LINES_REPLACED	0xdf	N	N	N	1	F	L3 Cache Lines Replaced
L2D_OZQ_CANCELS0	0xe0	Y	Y	Y	4	F	L2D OZQ cancels (Set 0)
L2D_OZQ_CANCELS1	0xe2	Y	Y	Y	4	F	L2D OZQ cancels (Set 1)
L2D_OZQ_FULL	0xe1 0xe3	N	N	N	1	F	L2D OZQ is full
L2D_BYPASS	0xe4	Y	Y	Y	4	F	Count bypasses
L2D_OZQ_RELEASE	0xe5	N	N	N	1	F	Clocks with release ordering attribute existed in L2D OZQ
L2D_REFERENCES	0xe6	Y	Y	Y	4	F	Data read/write access to L2D
L2D_L3ACCESS_CANCEL	0xe8	Y	Y	Y	1	F	Canceled L3 accesses

Table 4-42. All Performance Monitors Ordered by Code (Sheet 7 of 7)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Type	Description
L2D_OZDB_FULL	0xe9	N	N	N	1	F	L2D OZ data buffer is full
L2D_FORCE_RECIRC	0xea	Y	Y	Y	4	F	Forced recirculates
L2D_ISSUED_RECIRC_OZQ_ACC	0xeb	Y	Y	Y	1	F	Counts number of attempted OZQ recirculates back to L1D
L2D_BAD_LINES_SELECTED	0xec	Y	Y	Y	4	F	Valid line replaced when invalid line is available
L2D_STORE_HIT_SHARED	0xed	Y	Y	Y	2	F	Store hit a shared line
TAGGED_L2D_RETURN_PORT	0xee	Y	Y	Y	1	F	Tagged L2D Return Ports 0-3
L2D_OZQ_ACQUIRE	0xef	N	N	N	1	F	Clocks with acquire ordering attribute existed in L2D OZQ
L2D_OPS_ISSUED	0xf0	Y	Y	Y	4	F	Different operations issued by L2D
L2D_FILLB_FULL	0xf1	N	N	N	1	F	L2D Fill buffer is full
L2D_FILL_MESI_STATE	0xf2	Y	Y	Y	1	F	MESI state of L2D fills
L2D_VICTIMB_FULL	0xf3	N	N	N	1	F	L2D victim buffer is full

4.15 Performance Monitor Event List

This section enumerates Montecito processor performance monitoring events.

NOTE: Events that can be constrained by an Instruction Address Range can only be constrained by **IBRP0** unless otherwise noted.

ALAT_CAPACITY_MISS

- **Title:** ALAT Entry Replaced
- **Category:** Instruction Execution **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0x58, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Provides information on the number of times an advanced load (`ld.a`, `ld.as`, `ldfp.a` or `ldfp.as`) or missing `ld.c.nc` displaced a valid entry in the ALAT which did not have the same register id or replaced the last one to two invalid entries.

Table 4-43. Unit Masks for ALAT_CAPACITY_MISS

Extension	PMC.umask [19:16]	Description
---	bxx00	(* nothing will be counted *)
INT	bxx01	only integer instructions
FP	bxx10	only floating-point instructions
ALL	bxx11	both integer and floating-point instructions

BACK_END_BUBBLE

- **Title:** Full Pipe Bubbles in Main Pipe
- **Category:** Stall Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x00, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of full-pipe bubbles in the main pipe stalled due to any of 5 possible events (FPU/L1D, RSE, EXE, branch/exception or the front-end). One event unit mask further constrains this event and allows for some details in order to facilitate collecting all information with four counters.
- **NOTE:** During a thread switch, a banked counter will encounter a single "dead" cycle before being placed into the background with the thread it belongs to. If monitored in a **banked** counter, and this event occurs during that "dead" cycle, the event will be dropped. For this reason, monitoring a .all version of this event may not quite add to the component .me's.

Table 4-44. Unit Masks for BACK_END_BUBBLE

Extension	PMC.umask [19:16]	Description
ALL	bxx00	Front-end, RSE, EXE, FPU/L1D stall or a pipeline flush due to an exception/branch misprediction
FE	bxx01	front-end
L1D_FPU_RSE	bxx10	
---	bxx11	(* nothing will be counted *)

BE_BR_MISPRED_DETAIL

- **Title:** Back-end Branch Misprediction Detail
- **Category:** Branch Events **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x61, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of branches retired based on the prediction result, Back-end mispredictions of stg, rot, or pfs. These predictions are per bundle rather than per branch.
- **NOTE:** These events are counted only if there are no path mispredictions associated with branches because path misprediction guarantees stg/rot/pfs misprediction.

Table 4-45. Unit Masks for BE_BR_MISPREDICT_DETAIL

Extension	PMC.umask [19:16]	Description
ANY	bxx00	any back-end mispredictions
STG	bxx01	only back-end stage mispredictions
ROT	bxx10	only back-end rotate mispredictions
PFS	bxx11	only back-end pfs mispredictions for taken branches

BE_EXE_BUBBLE

- **Title:** Full Pipe Bubbles in Main Pipe due to Execution Unit Stalls
- **Category:** Stall Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x02, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of full-pipe bubbles in the main pipe due to stalls caused by the Execution Unit.
- **NOTE:** The different causes for this event are not prioritized because there is no need to do so (causes are independent and several of them fire at the same time, they all should be counted).

Table 4-46. Unit Masks for BE_EXE_BUBBLE

Extension	PMC.umask [19:16]	Description
ALL	b0000	was stalled by exe
GRALL	b0001	Back-end was stalled by exe due to GR/GR or GR/load dependency
FRALL	b0010	Back-end was stalled by exe due to FR/FR or FR/load dependency
PR	b0011	Back-end was stalled by exe due to PR dependency
ARCR	b0100	Back-end was stalled by exe due to AR or CR dependency
GRGR	b0101	Back-end was stalled by exe due to GR/GR dependency
CANCEL	b0110	Back-end was stalled by exe due to a canceled load
BANK_SWITCH	b0111	Back-end was stalled by exe due to bank switching.
ARCR_PR_CANCEL_BANK	b1000	ARCR, PR, CANCEL or BANK_SWITCH
---	b1001-b1111	(* nothing will be counted *)

BE_FLUSH_BUBBLE

- **Title:** Full Pipe Bubbles in Main Pipe due to Flushes.
- **Category:** Stall Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x04, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of full-pipe bubbles in the main pipe due to flushes.
- **NOTE:** XPN is higher priority than BRU. During a thread switch, a banked counter will encounter a single “dead” cycle before being placed into the background with the thread it belongs to. If monitored in a **banked** counter, and this event occurs during that “dead” cycle, the event will be dropped. For this reason, monitoring a .all version of this event may not quite add to the component .me’s.

Table 4-47. Unit Masks for BE_FLUSH_BUBBLE

Extension	PMC.umask [19:16]	Description
ALL	bxx00	Back-end was stalled due to either an exception/interruption or branch misprediction flush
BRU	bxx01	Back-end was stalled due to a branch misprediction flush

Table 4-47. Unit Masks for BE_FLUSH_BUBBLE

Extension	PMC.umask [19:16]	Description
XPN	bxx10	Back-end was stalled due to an exception/interruption flush This would include flush cycles for thread switch.
---	bxx11	(* nothing will be counted *)

BE_L1D_FPU_BUBBLE

- **Title:** Full Pipe Bubbles in Main Pipe due to FP or L1D Cache
- **Category:** Stall Events/L1D Cache Set 2 **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xca, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of full-pipe bubbles in the main pipe due to stalls caused by either floating-point unit or L1D cache.
- **NOTE:** This is a restricted set 2 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5. The different causes for this event are not prioritized because there is no need to do so (causes are independent and several of them fire at the same time, they all should be counted).

Table 4-48. Unit Masks for BE_L1D_FPU_BUBBLE (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
ALL	b0000	Back-end was stalled by L1D or FPU
FPU	b0001	Back-end was stalled by FPU.
L1D	b0010	Back-end was stalled by L1D. This includes all stalls caused by the L1 pipeline (created in the L1D stage of the L1 pipeline which corresponds to the DET stage of the main pipe).
L1D_FULLSTBUF	b0011	Back-end was stalled by L1D due to store buffer being full
L1D_PIPE_RECIRC	b0100	Back-end was stalled by L1D due a recirculate
L1D_HPW	b0101	Back-end was stalled by L1D due to Hardware Page Walker
---	b0110	(* count is undefined *)
L1D_FILLCONF	b0111	Back-end was stalled by L1D due a store in conflict with a returning fill.
L1D_AR_CR	b1000	Back-end was stalled by L1D due to ar/cr requiring a stall
L1D_L2BPRESS	b1001	Back-end was stalled by L1D due to L2D Back Pressure
L1D_TLB	b1010	Back-end was stalled by L1D due to L2DTLB to L1DTLB transfer
L1D_LDCONF	b1011	Back-end was stalled by L1D due to architectural ordering conflict
L1D_LDCHK	b1100	Back-end was stalled by L1D due to load check ordering conflict.
L1D_NAT	b1101	Back-end was stalled by L1D due to L1D data return needing recirculated NaT generation.

Table 4-48. Unit Masks for BE_L1D_FPU_BUBBLE (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
L1D_STBUFRECIR	b1110	Back-end was stalled by L1D due to store buffer cancel needing recirculate.
L1D_NATCONF	b1111	Back-end was stalled by L1D due to ld8.fill conflict with st8.spill not written to unat.

BE_LOST_BW_DUE_TO_FE

- **Title:** Invalid Bundles if BE Not Stalled for Other Reasons.
- **Category:** Stall Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x72, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of invalid bundles at the exit from Instruction Buffer only if Back-end is not stalled for other reasons.
- **NOTE:** Causes for lost bandwidth are prioritized in the following order from high to low for this event: FEFLUSH, TLBMISS, IMISS, PLP, BR_ILOCK, BRQ, BI, FILL_RECIRC, BUBBLE, IBFULL, UNREACHED. The prioritization implies that when several stall conditions exist at the same time, only the highest priority one will be counted. There are two cases where a bundle is considered “unreachable”. When bundle 0 contains a taken branch or bundle 0 is invalid but has IP[4] set to 1, bundle 1 will not be reached.

Table 4-49. Unit Masks for BE_LOST_BW_DUE_TO_FE

Extension	PMC.umask [19:16]	Description
ALL	b0000	count regardless of cause
FEFLUSH	b0001	only if caused by a front-end flush
---	b0010	(* illegal selection *)
---	b0011	(* illegal selection *)
UNREACHED	b0100	only if caused by unreachable bundle
IBFULL	b0101	(* meaningless for this event *)
IMISS	b0110	only if caused by instruction cache miss stall
TLBMISS	b0111	only if caused by TLB stall
FILL_RECIRC	b1000	only if caused by a recirculate for a cache line fill operation
BI	b1001	only if caused by branch initialization stall
BRQ	b1010	only if caused by branch retirement queue stall
PLP	b1011	only if caused by perfect loop prediction stall
BR_ILOCK	b1100	only if caused by branch interlock stall
BUBBLE	b1101	only if caused by branch resteer bubble stall
---	b1110-b1111	(* illegal selection *)

BE_RSE_BUBBLE

- **Title:** Full Pipe Bubbles in Main Pipe due to RSE Stalls
- **Category:** Stall Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x01, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of full-pipe bubbles in the main pipe due to stalls caused by the Register Stack Engine.
- **NOTE:** AR_DEP has a higher priority than OVERFLOW, UNDERFLOW and LOADRS. However, this is the only prioritization implemented. In order to count OVERFLOW, UNDERFLOW or LOADRS, AR_DEP must be false.

Table 4-50. Unit Masks for BE_RSE_BUBBLE

Extension	PMC.umask [19:16]	Description
ALL	bx000	Back-end was stalled by RSE
BANK_SWITCH	bx001	Back-end was stalled by RSE due to bank switching
AR_DEP	bx010	Back-end was stalled by RSE due to AR dependencies
OVERFLOW	bx011	Back-end was stalled by RSE due to need to spill
UNDERFLOW	bx100	Back-end was stalled by RSE due to need to fill
LOADRS	bx101	Back-end was stalled by RSE due to loadrs calculations
---	bx110-bx111	(* nothing will be counted *)

BR_MISPRED_DETAIL

- **Title:** FE Branch Mispredict Detail
- **Category:** Branch Events **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x5b, **Max. Inc/Cyc:** 3, **MT Capture Type:** A
- **Definition:** Counts the number of branches retired. All 16 values for PMC.umask are valid in order to provide information based on prediction result (mispredicted path or target address by front-end), and branch type.

Table 4-51. Unit Masks for BR_MISPRED_DETAIL

Extension	PMC.umask [19:16]	Description
ALL.ALL_PRED	b0000	All branch types, regardless of prediction result
ALL.CORRECT_PRED	b0001	All branch types, correctly predicted branches (outcome and target)
ALL.WRONG_PATH	b0010	All branch types, mispredicted branches due to wrong branch direction
ALL.WRONG_TARGET	b0011	All branch types, mispredicted branches due to wrong target for taken branches
IPREL.ALL_PRED	b0100	Only IP relative branches, regardless of prediction result
IPREL.CORRECT_PRED	b0101	Only IP relative branches, correctly predicted branches (outcome and target)
IPREL.WRONG_PATH	b0110	Only IP relative branches, mispredicted branches due to wrong branch direction

Table 4-51. Unit Masks for BR_MISPRED_DETAIL

Extension	PMC.umask [19:16]	Description
IPREL.WRONG_TARGET	b0111	Only IP relative branches, mispredicted branches due to wrong target for taken branches
RETURN.ALL_PRED	b1000	Only return type branches, regardless of prediction result
RETURN.CORRECT_PRED	b1001	Only return type branches, correctly predicted branches (outcome and target)
RETURN.WRONG_PATH	b1010	Only return type branches, mispredicted branches due to wrong branch direction
RETURN.WRONG_TARGET	b1011	Only return type branches, mispredicted branches due to wrong target for taken branches
NRETIND.ALL_PRED	b1100	Only non-return indirect branches, regardless of prediction result
NRETIND.CORRECT_PRED	b1101	Only non-return indirect branches, correctly predicted branches (outcome and target)
NRETIND.WRONG_PATH	b1110	Only non-return indirect branches, mispredicted branches due to wrong branch direction
NRETIND.WRONG_TARGET	b1111	Only non-return indirect branches, mispredicted branches due to wrong target for taken branches

BR_MISPRED_DETAIL2

- **Title:** FE Branch Mispredict Detail (Unknown Path Component)
- **Category:** Branch Events **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x68, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** This event goes with BR_MISPRED_DETAIL event based on prediction result and branch type
- **NOTE:** For accurate misprediction counts the following measurement must be taken:

  BR_MISPRED_DETAIL.[umask] - BR_MISPRED_DETAIL2.[umask]

  By performing this calculation for every umask, one can obtain a true value for the BR_MISPRED_DETAIL event.

Table 4-52. Unit Masks for BR_MISPREDICT_DETAIL2 (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
ALL.ALL_UNKNOWN_PRED	b0000	All branch types, branches with unknown path prediction
ALL.UNKNOWN_PATH_CORRECT_PRED	b0001	All branch types, branches with unknown path prediction and correctly predicted branch (outcome & target)
ALL.UNKNOWN_PATH_WRONG_PATH	b0010	All branch types, branches with unknown path prediction and wrong branch direction
---	b0011	(* nothing will be counted *)
IPREL.ALL_UNKNOWN_PRED	b0100	Only IP relative branches, branches with unknown path prediction
IPREL.UNKNOWN_PATH_CORRECT_PRED	b0101	Only IP relative branches, branches with unknown path prediction and correctly predicted branch (outcome & target)

Table 4-52. Unit Masks for BR_MISPREDICT_DETAIL2 (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
IPREL.UNKNOWN_PATH_WRONG_PATH	b0110	Only IP relative branches, branches with unknown path prediction and wrong branch direction
---	b0111	(* nothing will be counted *)
RETURN.ALL_UNKNOWN_PRED	b1000	Only return type branches, branches with unknown path prediction
RETURN.UNKNOWN_PATH_CORRECT_PRED	b1001	Only return type branches, branches with unknown path prediction and correctly predicted branch (outcome & target)
RETURN.UNKNOWN_PATH_WRONG_PATH	b1010	Only return type branches, branches with unknown path prediction and wrong branch direction
---	b1011	(* nothing will be counted *)
NRETIND.ALL_UNKNOWN_PRED	b1100	Only non-return indirect branches, branches with unknown path prediction
NRETIND.UNKNOWN_PATH_CORRECT_PRED	b1101	Only non-return indirect branches, branches with unknown path prediction and correctly predicted branch (outcome & target)
NRETIND.UNKNOWN_PATH_WRONG_PATH	b1110	Only non-return indirect branches, branches with unknown path prediction and wrong branch direction
---	b1111	(* nothing will be counted *)

BR_PATH_PRED

- **Title:** FE Branch Path Prediction Detail
- **Category:** Branch Events **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x54, **Max. Inc/Cyc:** 3, **MT Capture Type:** A
- **Definition:** Counts the number of branches retired based on branch direction (taken/not taken), branch predication and branch type. All 16 values for PMC.umask are valid.

Table 4-53. Unit Masks for BR_PATH_PRED (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
ALL.MISPRED_NOTTAKEN	b0000	All branch types, incorrectly predicted path and not taken branch
ALL.MISPRED_TAKEN	b0001	All branch types, incorrectly predicted path and taken branch
ALL.OKPRED_NOTTAKEN	b0010	All branch types, correctly predicted path and not taken branch
ALL.OKPRED_TAKEN	b0011	All branch types, correctly predicted path and taken branch
IPREL.MISPRED_NOTTAKEN	b0100	Only IP relative branches, incorrectly predicted path and not taken branch
IPREL.MISPRED_TAKEN	b0101	Only IP relative branches, incorrectly predicted path and taken branch
IPREL.OKPRED_NOTTAKEN	b0110	Only IP relative branches, correctly predicted path and not taken branch
IPREL.OKPRED_TAKEN	b0111	Only IP relative branches, correctly predicted path and taken branch

Table 4-53. Unit Masks for BR_PATH_PRED (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
RETURN.MISPRED_NOTTAKEN	b1000	Only return type branches, incorrectly predicted path and not taken branch
RETURN.MISPRED_TAKEN	b1001	Only return type branches, incorrectly predicted path and taken branch
RETURN.OKPRED_NOTTAKEN	b1010	Only return type branches, correctly predicted path and not taken branch
RETURN.OKPRED_TAKEN	b1011	Only return type branches, correctly predicted path and taken branch
NRETIND.MISPRED_NOTTAKEN	b1100	Only non-return indirect branches, incorrectly predicted path and not taken branch
NRETIND.MISPRED_TAKEN	b1101	Only non-return indirect branches, incorrectly predicted path and taken branch
NRETIND.OKPRED_NOTTAKEN	b1110	Only non-return indirect branches, correctly predicted path and not taken branch
NRETIND.OKPRED_TAKEN	b1111	Only non-return indirect branches, correctly predicted path and taken branch

BR_PATH_PRED2

- **Title:** FE Branch Path Prediction Detail (Unknown Pred Component)
- **Category:** Branch Events **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x6a, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** This event goes with BR_PATH_PREDICTION event.
- **NOTE:** When there is more than one branch in a bundle and one is predicted as taken, all the higher number ports are forced to a predicted not taken mode without actually knowing the their true prediction.

  The true OKPRED_NOTTAKEN predicted path information can be obtained by calculating:

  BR_PATH_PRED.[branch type].OKPRED_NOTTAKEN - BR_PATH_PRED2.[branch type].UNKNOWNPRED_NOTTAKEN using the same “branch type” (ALL, IPREL, RETURN, NRETIND) specified for both events.

  Similarly, the true MISPRED_TAKEN predicted path information can be obtained by calculating:

  BR_PATH_PRED.[branch type].MISPRED_TAKEN - BR_PATH_PRED2.[branch type].UKNOWNPRED_TAKEN using the same “branch type” (ALL, IPREL, RETURN, NRETIND) selected for both events.

Table 4-54. Unit Masks for BR_PATH_PRED2 (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
ALL.UNKNOWNPRED_NOTTAKEN	b00x0	All branch types, unknown predicted path and not taken branch (which impacts OKPRED_NOTTAKEN)
ALL.UNKNOWNPRED_TAKEN	b00x1	All branch types, unknown predicted path and taken branch (which impacts MISPRED_TAKEN)
IPREL.UNKNOWNPRED_NOTTAKEN	b01x0	Only IP relative branches, unknown predicted path and not taken branch (which impacts OKPRED_NOTTAKEN)

Table 4-54. Unit Masks for BR_PATH_PRED2 (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
IPREL.UNKNOWNPRED_TAKEN	b01x1	Only IP relative branches, unknown predicted path and taken branch (which impacts MISPRED_TAKEN)
RETURN.UNKNOWNPRED_NOTTAKEN	b10x0	Only return type branches, unknown predicted path and not taken branch (which impacts OKPRED_NOTTAKEN)
RETURN.UNKNOWNPRED_TAKEN	b10x1	Only return type branches, unknown predicted path and taken branch (which impacts MISPRED_TAKEN)
NRETIND.UNKNOWNPRED_NOTTAKEN	b11x0	Only non-return indirect branches, unknown predicted path and not taken branch (which impacts OKPRED_NOTTAKEN)
NRETIND.UNKNOWNPRED_TAKEN	b11x1	Only non-return indirect branches, unknown predicted path and taken branch (which impacts MISPRED_TAKEN)

BUS_ALL

- **Title:** Bus Transactions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x87, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of bus transactions.

Table 4-55. Unit Masks for BUS_ALL

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
IO	bxx01	transactions initiated by non-CPU priority agents
SELF	bxx10	transactions initiated by ‘this’ cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_B2B_DATA_CYCLES

- **Title:** Back to Back Data Cycles on the Bus
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x93, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Counts the number of BUS Clocks which had valid data cycle driven back-to-back on the bus.

Table 4-56. Unit Masks for BUS_B2B_DATA_CYCLES

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
IO	bxx01	transactions initiated by non-CPU priority agents
SELF	bxx10	transactions initiated by 'this' cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_DATA_CYCLE

- **Title:** Valid Data Cycle on the Bus
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x88, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Counts the number of BUS Clocks which had a valid data cycle on the bus.

Table 4-57. Unit Masks for BUS_DATA_CYCLE

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
IO	bxx01	transactions initiated by non-CPU priority agents
SELF	bxx10	transactions initiated by 'this' cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_HITM

- **Title:** Bus Hit Modified Line Transactions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x84, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of transactions with HITM asserted (i.e. transaction was satisfied by some other processor's modified line).
- **NOTE:** This is equivalent to: BUS_RD_INVAL_BST_HITM + BUS_RD_HITM

Table 4-58. Unit Masks for BUS_HITM

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
IO	bxx01	transactions initiated by non-CPU priority agents
SELF	bxx10	transactions initiated by 'this' cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_IO

- **Title:** IA-32 Compatible IO Bus Transactions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x90, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of IA-32 I/O transactions.

Table 4-59. Unit Masks for BUS_IO

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
IO	bxx01	transactions initiated by non-CPU priority agents
SELF	bxx10	transactions initiated by 'this' cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_MEMORY

- **Title:** Bus Memory Transactions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x8a, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of bus memory transactions (i.e memory-read-invalidate, reserved-memory-read, memory-read, and memory-write transactions).

Table 4-60. Unit Masks for BUS_MEMORY

Extension	PMC.umask [19:16]	Description
---	b00xx	(* nothing will be counted *)
EQ_128BYTE.EITHER	b0100	number of full cache line transactions (BRL, BRIL, BWL, BRC, BCR, BCCL) from either local processor
EQ_128BYTE.IO	b0101	number of full cache line transactions (BRL, BRIL, BWL, BRC, BCR, BCCL) from non-CPU priority agents
EQ_128BYTE.SELF	b0110	number of full cache line transactions (BRL, BRIL, BWL, BRC, BCR, BCCL) from 'this' processor
EQ_128BYTE.ANY	b0111	number of full cache line transactions (BRL, BRIL, BWL, BRC, BCR, BCCL) from CPU or non-CPU (all transactions).
LT_128BYTE.EITHER	b1000	number of less than full cache line transactions (BRP, BWP, BIL) from either local processor
LT_128BYTE.IO	b1001	number of less than full cache line transactions (BRP, BWP, BIL) from non-CPU priority agents
LT_128BYTE.SELF	b1010	number of less than full cache line transactions (BRP, BWP, BIL) from 'this' processor
LT_128BYTE.ANY	b1011	number of less than full cache line transactions (BRP, BWP, BIL) CPU or non-CPU (all transactions).
ALL.EITHER	b1100	All bus transactions from either local processor
ALL.IO	b1101	All bus transactions from non-CPU priority agents

Table 4-60. Unit Masks for BUS_MEMORY

Extension	PMC.umask [19:16]	Description
ALL.SELF	b1110	All bus transactions from 'this' local processor
ALL.ANY	b1111	All bus transactions from CPU or non-CPU (all transactions).

BUS_MEM_READ

- **Title:** Full Cache Line D/I Memory RD, RD Invalidate, and BRIL
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x8b, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of full cache-line (128-byte) data/code memory read (BRL), full cache-line memory read-invalidate (BRIL), and 0-byte memory read-invalidate (BIL) transactions.

Table 4-61. Unit Masks for BUS_MEM_READ

Extension	PMC.umask [19:16]	Description
BIL.EITHER	b0000	Number of BIL 0-byte memory read invalidate transactions from either local processor
BIL.IO	b0001	Number of BIL 0-byte memory read invalidate transactions from non-CPU priority agents
BIL.SELF	b0010	Number of BIL 0-byte memory read invalidate transactions from 'this' processor
BIL.ANY	b0011	Number of BIL 0-byte memory read invalidate transactions from CPU or non-CPU (all transactions).
BRL.EITHER	b0100	Number of full cache line memory read transactions from either local processor
BRL.IO	b0101	Number of full cache line memory read transactions from non-CPU priority agents
BRL.SELF	b0110	Number of full cache line memory read transactions from 'this' processor
BRL.ANY	b0111	Number of full cache line memory read transactions from CPU or non-CPU (all transactions).
BRIL.EITHER	b1000	Number of full cache line memory read invalidate transactions from either local processor
BRIL.IO	b1001	Number of full cache line memory read invalidate transactions from non-CPU priority agents
BRIL.SELF	b1010	Number of full cache line memory read invalidate transactions from 'this'processor
BRIL.ANY	b1011	Number of full cache line memory read invalidate transactions from CPU or non-CPU (all transactions).
ALL.EITHER	b1100	All memory read transactions from either local processor
ALL.IO	b1101	All memory read transactions from non-CPU priority agents
ALL.SELF	b1110	All memory read transactions from 'this'processor
ALL.ANY	b1111	All memory read transactions from CPU or non-CPU (all transactions).

BUS_RD_DATA

- **Title:** Bus Read Data Transactions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x8c, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of full-cache-line (128-byte) data memory read transactions (BRL).

Table 4-62. Unit Masks for BUS_RD_DATA

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
IO	bxx01	transactions initiated by non-CPU priority agents
SELF	bxx10	transactions initiated by 'this' cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_RD_HIT

- **Title:** Bus Read Hit Clean Non-local Cache Transactions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x80, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of bus reads that hit a clean line in another processor's cache (implies HIT and BRL).

Table 4-63. Unit Masks for BUS_RD_HIT

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
IO	bxx01	transactions initiated by non-CPU priority agents
SELF	bxx10	transactions initiated by 'this' cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_RD_HITM

- **Title:** Bus Read Hit Modified Non-local Cache Transactions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x81, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of bus reads that hit a modified line in another processor's cache (implies HITM and BRL).

Table 4-64. Unit Masks for BUS_RD_HITM

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
IO	bxx01	transactions initiated by non-CPU priority agents
SELF	bxx10	transactions initiated by 'this' cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_RD_INVAL_BST_HITM

- **Title:** Bus BRIL Transaction Results in HITM
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x83, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of bus read invalidate line transactions (implies BRIL and HITM) which are satisfied from a remote processor only.

Table 4-65. Unit Masks for BUS_RD_INVAL_BST_HITM

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
IO	bxx01	transactions initiated by non-CPU priority agents
SELF	bxx10	transactions initiated by 'this' cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_RD_INVAL_HITM

- **Title:** Bus BIL Transaction Results in HITM
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x82, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of bus read invalidated line transactions for which HITM was asserted (implies BIL and HITM) and the transaction was satisfied from another processor's cache.

Table 4-66. Unit Masks for BUS_RD_INVAL_HITM

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
IO	bxx01	transactions initiated by non-CPU priority agents
SELF	bxx10	transactions initiated by 'this' cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_RD_IO

- **Title:** IA-32 Compatible IO Read Transactions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x91, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of IA-32 I/O read transactions.

Table 4-67. Unit Masks for BUS_RD_IO

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
IO	bxx01	transactions initiated by non-CPU priority agents
SELF	bxx10	transactions initiated by 'this' cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_RD_PRTL

- **Title:** Bus Read Partial Transactions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x8d, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of less-than-full-cache-line (0,8,16,32, and 64 byte) memory read transactions (BRP).

Table 4-68. Unit Masks for BUS_RD_PRTL

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
IO	bxx01	transactions initiated by non-CPU priority agents
SELF	bxx10	transactions initiated by 'this' cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_SNOOP_STALL_CYCLES

- **Title:** Bus Snoop Stall Cycles (from any agent)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x8f, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of bus clocks FSB is stalled for snoop.

Table 4-69. Unit Masks for BUS_SNOOP_STALL_CYCLES

Extension	PMC.umask [19:16]	Description
EITHER	bxx00	transactions initiated by either cpu core
---	bxx01	(* illegal selection *)
SELF	bxx10	transactions initiated by 'this' cpu core
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_WR_WB

- **Title:** Bus Write Back Transactions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x92, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of write-back memory write transactions (BWL writes due to M-state line write-backs and coalesced writes).

Table 4-70. Unit Masks for BUS_WR_WB (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
---	b00xx	(* nothing will be counted *)
EQ_128BYTE.EITHER	b0100	either local processor/Only cache line transactions with write back or write coalesce attributes will be counted.
EQ_128BYTE.IO	b0101	non-CPU priority agents/Only cache line transactions with write back or write coalesce attributes will be counted.
EQ_128BYTE.SELF	b0110	'this' processor/Only cache line transactions with write back or write coalesce attributes will be counted.
EQ_128BYTE.ANY	b0111	CPU or non-CPU (all transactions)./Only cache line transactions with write back or write coalesce attributes will be counted.
CCASTOUT.EITHER	b1000	either local processor/Only 0-byte transactions with write back attribute (clean cast outs) will be counted
---	b1001	(* illegal selection *)
CCASTOUT.SELF	b1010	'this' processor/Only 0-byte transactions with write back attribute (clean cast outs) will be counted
CCASTOUT.ANY	b1011	CPU or non-CPU (all transactions)/Only 0-byte transactions with write back attribute (clean cast outs) will be counted
ALL.EITHER	b1100	either local processor
ALL.IO	b1101	non-CPU priority agents

Table 4-70. Unit Masks for BUS_WR_WB (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
ALL.SELF	b1110	'this' processor
ALL.ANY	b1111	CPU or non-CPU (all transactions).

CPU_CPL_CHANGES

- **Title:** Privilege Level Changes
- **Category:** System Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x13, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of privilege level changes.

Table 4-71. Unit Masks for CPU_CPL_CHANGES

Extension	PMC.umask [19:16]	Description
---	b0000	(* nothing will be counted *)
LVL0	b0001	All changes to/from privilege level 0 are counted
LVL1	b0010	All changes to/from privilege level 1 are counted
LVL2	b0100	All changes to/from privilege level 2 are counted
LVL3	b1000	All changes to/from privilege level 3 are counted
ALL	b1111	All changes in cpl counted

CPU_OP_CYCLES

- **Title:** CPU Operating Cycles
- **Category:** Basic Events **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x12, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Counts the number of core clock cycles. This event does not count cycles when the thread is in low power mode. This event has a umask which allows counting only the cycles when processing qualified instructions. Instruction can be qualified using regular address range checking and/or opcode match qualification. Further, it is necessary to program channel 1 to count all instructions without any qualifications.
- **NOTE:** Although CPU_OP_CYCLES{all} is supported and expected to increment as long as either of the threads are executing, CPU_OP_CYCLES{all} will not increment when the thread the counter register belongs to enters a low-power halt if the other thread is not also in a low-power halt state. When threads are expected to enter a low-power halt state, it is expected that software will add the CPU_OP_CYCLES{me} for each thread in order to calculate CPU_OP_CYCLES{all}. If CPU_OP_CYCLES are being captured in a **banked** counter and *.all* is enabled, a single cycle will be dropped from the count each time a thread switch occurs. It is possible to compensate by monitoring THREAD_SWITCH_EVENTS.ALL and adding the two counts together.

Table 4-72. Unit Masks for CPU_OP_CYCLES

Extension	PMC.umask [16]	Description
ALL	bxxx0	All CPU cycles counted
QUAL	bxxx1	Qualified cycles only

CPU_OP_CYCLES_HALTED

- **Title:** CPU Operating Cycles Halted
- **Category:** System Event **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x18, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Counts the number of core clock cycles the thread is BAhalted.
- **NOTE:** Only PMC/D10 is capable of monitoring this event.

DATA_DEBUG_REGISTER_FAULT

- **Title:** Fault Due to Data Debug Reg. Match to Load/Store Instruction
- **Category:** System Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x52, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of times we take a fault due to one of data debug registers matching a load or store instruction.

DATA_DEBUG_REGISTER_MATCHES

- **Title:** Data Debug Register Matches Data Address of Memory References.
- **Category:** System Events **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xc6, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of times the data debug register matches the data address of a memory reference. This is the OR function the 4 DBR matches. Registers DBR0-7, PSR, DCR, PMC13 affect this event. It does not include commits which means that it might have noise.

DATA_EAR_EVENTS

- **Title:** L1 Data Cache EAR Events
- **Category:** L1 Data Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xc8, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of L1 Data Cache or L1DTLB or ALAT events captured by EAR

DATA_REFERENCES_SET0

- **Title:** Data Memory References Issued to Memory Pipeline
- **Category:** L1 Data Cache/L1D Cache Set 0 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xc3, **Max. Inc/Cyc:** 4, **MT Capture Type:** A
- **Definition:** Counts the number of data memory references issued into memory pipeline (includes check loads, uncacheable accesses, RSE operations, semaphores, and floating-point memory references). The count includes wrong path operations but excludes predicated off operations. This event does not include VHPT memory references.
- **NOTE:** This is a restricted set 0 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5.

DATA_REFERENCES_SET1

- **Title:** Data Memory References Issued to Memory Pipeline
- **Category:** L1 Data Cache/L1D Cache Set 1 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xc5, **Max. Inc/Cyc:** 4, **MT Capture Type:** A
- **Definition:** Counts the number of data memory references issued into memory pipeline (includes check loads, uncacheable accesses, RSE operations, semaphores, and floating-point memory references). The count includes wrong path operations but excludes predicated off operations. This event does not include VHPT memory references.
- **NOTE:** This is a restricted set 1 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5.

DISP_STALLED

- **Title:** Number of Cycles Dispersal Stalled
- **Category:** Instruction Dispersal Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x49, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of cycles dispersal was stalled due to flushes or back-end pipeline stalls.

DTLB_INSERTS_HPW

- **Title:** Hardware Page Walker Inserts to DTLB
- **Category:** TLB **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xc9, **Max. Inc/Cyc:** 4, **MT Capture Type:** F
- **Definition:** Counts the number of VHPT entries inserted into DTLB by Hardware Page Walker.
- **NOTE:** This will include misses which the DTLB did not squash even though the instructions causing the miss did not get to retirement.

ENCBR_MISPRED_DETAIL

- **Title:** Number of Encoded Branches Retired
- **Category:** Branch Events **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x63, **Max. Inc/Cyc:** 3, **MT Capture Type:** A
- **Definition:** Counts the number of branches retired only if there is a branch on port B0 (i.e. encoded branch).

Table 4-73. Unit Masks for ENCBR_MISPRED_DETAIL

Extension	PMC.umask [19:16]	Description
ALL.ALL_PRED	b0000	all encoded branches, regardless of prediction result
ALL.CORRECT_PRED	b0001	all encoded branches, correctly predicted branches (outcome and target)
ALL.WRONG_PATH	b0010	all encoded branches, mispredicted branches due to wrong branch direction
ALL.WRONG_TARGET	b0011	all encoded branches, mispredicted branches due to wrong target for taken branches
---	b0100	(* nothing will be counted *)
---	b0101	(* nothing will be counted *)
---	b0110	(* nothing will be counted *)
---	b0111	(* nothing will be counted *)
OVERSUB.ALL_PRED	b1000	only those which cause oversubscription, regardless of prediction result
OVERSUB.CORRECT_PRED	b1001	only those which cause oversubscription, correctly predicted branches (outcome and target)
OVERSUB.WRONG_PATH	b1010	only those which cause oversubscription, mispredicted branches due to wrong branch direction
OVERSUB.WRONG_TARGET	b1011	only those which cause oversubscription mispredicted branches due to wrong target for taken branches
ALL2.ALL_PRED	b1100	all encoded branches, regardless of prediction result
ALL2.CORRECT_PRED	b1101	all encoded branches, correctly predicted branches (outcome and target)
ALL2.WRONG_PATH	b1110	all encoded branches, mispredicted branches due to wrong branch direction
ALL2.WRONG_TARGET	b1111	all encoded branches, mispredicted branches due to wrong target for taken branches

ER_BKSNP_ME_ACCEPTED

- **Title:** Backsnoop Me Accepted
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xbb, **Max. Inc/Cyc:** 2, **MT Capture Type:** C
- **Definition:** Counts the number of BacksnoopMe requests accepted into the BRQ from the L2d (used by the L2d to get itself out of potential forward progress situations).
- **NOTE:** The *.all* field is ignored for this event. The event will count as if *.all* is set.

ER_BRQ_LIVE_REQ_HI

- **Title:** BRQ Live Requests (upper two bits)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xb8, **Max. Inc/Cyc:** 2, **MT Capture Type:** C
- **Definition:** Counts the number of live read requests in BRQ. The Montecito processor can have a total of 16 per cycle. The upper 2 bits are stored in this counter (bits 4:3).
- **NOTE:** If a read request has an L2d victim, it is also entered in the BRQ (as writeback). This event will count 1 as long as a read or its victim is in BRQ (net effect is that due to an L2d victim, the life of read in BRQ is extended). The *.all* field is ignored for this event. The event will count as if *.all* is set.

ER_BRQ_LIVE_REQ_LO

- **Title:** BRQ Live Requests (lower three bits)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xb9, **Max. Inc/Cyc:** 7, **MT Capture Type:** C
- **Definition:** Counts the number of live read requests in BRQ. The Montecito processor can have a total of 16 per cycle. The lower 3 bits are stored in this counter (bits 2:0).
- **NOTE:** If a read request has an L2d victim, it is also entered in the BRQ (as writeback). This event will count 1 as long as a read or its victim is in BRQ (net effect is that due to an L2d victim, the life of read in BRQ is extended). The *.all* field is ignored for this event. The event will count as if *.all* is set.

ER_BRQ_REQ_INSERTED

- **Title:** BRQ Requests Inserted
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xba, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of requests which are inserted into BRQ.
- **NOTE:** Entries made into BRQ due to L2d victims (caused by read, fc, cc) are not counted.

ER_MEM_READ_OUT_HI

- **Title:** Outstanding Memory Read Transactions (upper 2 bits)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xb4, **Max. Inc/Cyc:** 2, **MT Capture Type:** F
- **Definition:** Counts the number of memory read transactions outstanding. The Montecito processor can have a total of 16 of this event per cycle. The upper two bits are stored in this counter. For the purpose of this event, a memory read access is assumed outstanding from the time a read request is issued on the FSB until the first chunk of read data is returned to L2D.
- **NOTE:** Uncacheables (or anything else which doesn't access the L3) are not tracked. This is intended to be used along with BUS_MEM_READ for average system memory latency

ER_MEM_READ_OUT_LO

- **Title:** Outstanding Memory Read Transactions (lower 3 bits)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xb5, **Max. Inc/Cyc:** 7, **MT Capture Type:** F
- **Definition:** Counts the number of memory read transactions outstanding. The Itanium 2 processor can have a total of 16 of this event per cycle. The lower three bits are stored in this counter. For the purpose of this event, a memory read access is assumed outstanding from the time a read request is issued on the FSB until the first chunk of read data is returned to L2D.
- **NOTE:** Uncacheables (or anything else which doesn't access the L3) are not tracked. This is intended to be used along with BUS_MEM_READ for average system memory latency

ER_REJECT_ALL_L1_REQ

- **Title:** Reject All L1 Requests
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xbc, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Number of cycles in which the BRQ was rejecting all L1i/L1d requests (for the “Big Hammer” forward progress logic).
- **NOTE:** The *.all* field is ignored for this event. The event will count as if *.all* is set.

ER_REJECT_ALL_L1D_REQ

- **Title:** Reject All L1D Requests
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xbd, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Number of cycles in which the BRQ was rejecting all L1D requests (for L1D/L1I forward progress).
- **NOTE:** The *.all* field is ignored for this event. The event will count as if *.all* is set.

ER_REJECT_ALL_L1I_REQ

- **Title:** Reject All L1I Requests
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xbe, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Number of cycles in which the BRQ was rejecting all L1I requests (for L1D/L1I forward progress).
- **NOTE:** The *.all* field is ignored for this event. The event will count as if *.all* is set.

ER_SNOOPQ_REQ_HI

- **Title:** Outstanding Snoops (upper bit)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xb6, **Max. Inc/Cyc:** 2, **MT Capture Type:** C
- **Definition:** Counts the number of snoops outstanding. The Montecito processor can have a total of 8 of this event per cycle. The upper bit is stored in this counter.
- **NOTE:** The *.all* field is ignored for this event. The event will count as if *.all* is set.

ER_SNOOPQ_REQ_LO

- **Title:** Outstanding Snoops (lower 3 bits)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xb7, **Max. Inc/Cyc:** 7, **MT Capture Type:** C
- **Definition:** Counts the number of memory read transactions outstanding. The Montecito processor can have a total of 8 of this event per cycle. The lower three bits are stored in this counter.
- **NOTE:** The *.all* field is ignored for this event. The event will count as if *.all* is set.

ETB_EVENT

- **Title:** Execution Trace Buffer Event Captured
- **Category:** Branch Events **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x11, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of entries captured in Execution Trace Buffer. Please see Section 3.3.10.1.1 and Section 3.3.10.2 for more information. Entries captured are subject to the constraints programmed to PMC_{39}.

FE_BUBBLE

- **Title:** Bubbles Seen by FE
- **Category:** Stall Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x71, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of bubbles seen by front-end. This event is another way of looking at the FE_LOST_BW event.

  Causes for stall are prioritized in the following order from high to low for this event: FEFLUSH, TLBMISS, IMISS, BRANCH, FILL_RECIRC, BUBBLE, IBFULL. The prioritization implies that when several stall conditions exist at the same time, only the highest priority one will be counted.

Table 4-74. Unit Masks for FE_BUBBLE (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
ALL	b0000	count regardless of cause
FEFLUSH	b0001	only if caused by a front-end flush
---	b0010	(*illegal selection*)
GROUP1	b0011	BUBBLE or BRANCH
GROUP2	b0100	IMISS or TLBMISS
IBFULL	b0101	only if caused by instruction buffer full stall
IMISS	b0110	only if caused by instruction cache miss stall
TLBMISS	b0111	only if caused by TLB stall
FILL_RECIRC	b1000	only if caused by a recirculate for a fill operation
BRANCH	b1001	only if caused by any of 4 branch recirculates
GROUP3	b1010	FILL_RECIRC or BRANCH
ALLBUT_FEFLUSH_BUBBLE	b1011	ALL except FEFLUSH and BUBBLE
ALLBUT_IBFULL	b1100	ALL except IBFULL

Table 4-74. Unit Masks for FE_BUBBLE (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
BUBBLE	b1101	only if caused by branch bubble stall
---	b1110-b1111	(* illegal selection *)

FE_LOST_BW

- **Title:** Invalid Bundles at the Entrance to IB
- **Category:** Stall Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x70, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of invalid bundles at the entrance to Instruction Buffer.
- **NOTE:** Causes for lost bandwidth are prioritized in the following order from high to low for this event: FEFLUSH, TLBMISS, IMISS, PLP, BR_ILOCK, BRQ, BI, FILL_RECIRC, BUBBLE, IBFULL, UNREACHED. The prioritization implies that when several stall conditions exist at the same time, only the highest priority one will be counted. There are two cases where a bundle is considered “unreachable”. When bundle 0 contains a taken branch or bundle 0 is invalid but has IP[4] set to 1, bundle 1 will not be reached.

Table 4-75. Unit Masks for FE_LOST_BW

Extension	PMC.umask [19:16]	Description
ALL	b0000	count regardless of cause
FEFLUSH	b0001	only if caused by a front-end flush
---	b0010	(* illegal selection *)
---	b0011	(* illegal selection *)
UNREACHED	b0100	only if caused by unreachable bundle
IBFULL	b0101	only if caused by instruction buffer full stall
IMISS	b0110	only if caused by instruction cache miss stall
TLBMISS	b0111	only if caused by TLB stall
FILL_RECIRC	b1000	only if caused by a recirculate for a cache line fill operation
BI	b1001	only if caused by branch initialization stall
BRQ	b1010	only if caused by branch retirement queue stall
PLP	b1011	only if caused by perfect loop prediction stall
BR_ILOCK	b1100	only if caused by branch interlock stall
BUBBLE	b1101	only if caused by branch resteer bubble stall
---	b1101-b1111	(* illegal selection *)

FP_FAILED_FCHKF

- **Title:** Failed fchkf
- **Category:** Instruction Execution **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x06, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of times the fchkf instruction failed.

FP_FALSE_SIRSTALL

- **Title:** SIR Stall Without a Trap
- **Category:** Instruction Execution **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x05, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of times SIR (Safe Instruction Recognition) stall is asserted and does not lead to a trap.

FP_FLUSH_TO_ZERO

- **Title:** FP Result Flushed to Zero
- **Category:** Instruction Execution **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x0b, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of times a near zero result is flushed to zero in FTZ mode. This has the following umasks.

Table 4-76. Unit Masks for FP_FLUSH_TO_ZERO

Extension	PMC.umask [16]	Description
FTZ_Real	b0	Times FTZ occured
FTZ_Poss	b1	Times FTZ would have occured if FTZ were enabled

FP_OPS_RETIRED

- **Title:** Retired FP Operations
- **Category:** Instruction Execution **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x09, **Max. Inc/Cyc:** 6, **MT Capture Type:** A
- **Definition:** Provides information on number of retired floating-point operations, excluding all predicated off instructions. This is a weighted sum of basic floating-point operations. To count how often specific opcodes are retired, use IA64_TAGGED_INST_RETIRED.
- **NOTE:** The following weights are used:

  Counted as 4 ops: fpma, fpms, and fpnma

  Counted as 2 ops: fpma, fpnma (f2=f0), fma, fms, fnma, fprcpa, fprsqrta, fpmpy, fpmax, fpamin, fpamax, fpcmp, fpcvt

  Counted as 1 op: fms, fma, fnma (f2=f0 or f4=f1), fmpy, fadd, fsub, frcpa, frsqrta, fmin, fmax, famin, famax, fpmin, fcvt.fx, `fcmp`

FP_TRUE_SIRSTALL

- **Title:** SIR Stall Asserted and Leads to a Trap
- **Category:** Instruction Execution **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x03, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of times SIR (Safe Instruction Recognition) stall is asserted and leads to a trap.

HPW_DATA_REFERENCES

- **Title:** Data Memory References to VHPT
- **Category:** L1 Data Cache **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x2d, **Max. Inc/Cyc:** 4, **MT Capture Type:** A
- **Definition:** Counts the number of data memory references to VHPT.
- **NOTE:** This will include misses the L2DTLB did not squash even though the instructions causing the miss did not get to retirement. HPW references originating very close to an ongoing thread switch may or may not be counted.

IA64_INST_RETIRED

- **Title:** Retired Itanium Instructions
- **Category:** Basic Events **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x08, **Max. Inc/Cyc:** 6, **MT Capture Type:** A
- **Definition:** Counts the number of retired instructions excluding hardware generated RSE operations and instructions. This event includes all retired instructions including predicated off instructions and nop instructions. This is a sub event of IA64_TAGGED_INST_RETIRED.
- **NOTE:** MLX bundles will be counted as no more than two instructions. Make sure that the corresponding registers are setup such that nothing will be constrained by the IBRP-PMC combination of interest (power up default is no constraints).

Table 4-77. Unit Masks for IA64_INST_RETIRED

Extension	PMC.umask [19:16]	Description
THIS	bxx00	Retired Itanium® Instructions

IA64_TAGGED_INST_RETIRED

- **Title:** Retired Tagged Instructions
- **Category:** Instruction Execution **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x08, **Max. Inc/Cyc:** 6, **MT Capture Type:** A
- **Definition:** Counts the number of retired instructions, excluding hardware generated RSE operations, that match the Instruction Address Breakpoint (IBRs) and Opcode Match register settings (PMC32,33,34,35). This event includes all instructions which reached retirement (including predicated off instructions and nop instructions). See Chapter 3 for more details about how to program different registers.
- **NOTE:** MLX bundles will be counted as no more than two instructions.

Table 4-78. Unit Masks for IA64_TAGGED_INST_RETIRED

Extension	PMC.umask [19:16]	Description
IBRP0_PMC32_33	bxx00	Instruction tagged by Instruction Breakpoint Pair 0 and the opcode matcher pair PMC32 and PMC33.
IBRP1_PMC34_35	bxx01	Instruction tagged by Instruction Breakpoint Pair 1 and the opcode matcher pair PMC34 and PMC35.
IBRP2_PMC32_33	bxx10	Instruction tagged by Instruction Breakpoint Pair 2 and the opcode matcher pair PMC32 and PMC33.
IBRP3_PMC34_35	bxx11	Instruction tagged by Instruction Breakpoint Pair 3 and the opcode matcher pair PMC34 and PMC35.

IDEAL_BE_LOST_BW_DUE_TO_FE

- **Title:** Invalid Bundles at the Exit From IB
- **Category:** Stall Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x73, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of invalid bundles at the exit from Instruction Buffer regardless of whether Back-end is stalled for other reasons or not.
- **NOTE:** Causes for lost bandwidth are prioritized in the following order from high to low for this event: FEFLUSH, TLBMISS, IMISS, PLP, BR_ILOCK, BRQ, BI, FILL_RECIRC, BUBBLE, IBFULL, UNREACHED. The prioritization implies that when several stall conditions exist at the same time, only the highest priority one will be counted. There are two cases where a bundle is considered “unreachable”. When bundle 0 contains a taken branch or bundle 0 is invalid but has IP[4] set to 1, bundle 1 will not be reached.

Table 4-79. Unit Masks for IDEAL_BE_LOST_BW_DUE_TO_FE

Extension	PMC.umask [19:16]	Description
ALL	b0000	count regardless of cause
FEFLUSH	b0001	only if caused by a front-end flush
---	b0010	(* illegal selection *)
---	b0011	(* illegal selection *)
UNREACHED	b0100	only if caused by unreachable bundle
IBFULL	b0101	(* meaningless for this event *)
IMISS	b0110	only if caused by instruction cache miss stall
TLBMISS	b0111	only if caused by TLB stall
FILL_RECIRC	b1000	only if caused by a recirculate for a cache line fill operation
BI	b1001	only if caused by branch initialization stall
BRQ	b1010	only if caused by branch retirement queue stall
PLP	b1011	only if caused by perfect loop prediction stall
BR_ILOCK	b1100	only if caused by branch interlock stall
BUBBLE	b1101	only if caused by branch resteer bubble stall
---	b1101-b1111	(* illegal selection *)

INST_CHKA_LDC_ALAT

- **Title:** Retired `chk.a` and `ld.c` Instructions
- **Category:** Instruction Execution **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0x56, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Provides information on the number of all advanced check load (`chk.a`) and check load (`ld.c`) instructions that reach retirement.
- **NOTE:** Faulting `chk.a` will be counted even if an older sibling faults.

Table 4-80. Unit Masks for INST_CHKA_LDC_ALAT

Extension	PMC.umask [19:16]	Description
---	bxx00	(* nothing will be counted *)
INT	bxx01	only integer instructions
FP	bxx10	only floating-point instructions
ALL	bxx11	both integer and floating-point instructions

INST_DISPERSED

- **Title:** Number of Syllables Dispersed from REN to REG
- **Category:** Instruction Dispersal Events **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x4d, **Max. Inc/Cyc:** 6, **MT Capture Type:** A
- **Definition:** Counts the number of syllables dispersed from REName to the REGister pipe stage in order to approximate those dispersed from ROTate to EXPand.

INST_FAILED_CHKA_LDC_ALAT

- **Title:** Failed `chk.a` and `ld.c` Instructions
- **Category:** Instruction Execution **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0x57, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Provides information on the number of failed advanced check load (`chk.a`) and check load (`ld.c`) instructions that reach retirement.
- **NOTE:** Although at any given time, there could be 2 failing `chk.a` or `ld.c`, only the first one is counted.

Table 4-81. Unit Masks for INST_FAILED_CHKA_LDC_ALAT

Extension	PMC.umask [19:16]	Description
---	bxx00	(* nothing will be counted *)
INT	bxx01	only integer instructions
FP	bxx10	only floating-point instructions
ALL	bxx11	both integer and floating-point instructions

INST_FAILED_CHKS_RETIRED

- **Title:** Failed `chk.s` Instructions
- **Category:** Instruction Execution **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x55, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Provides information on the number of failed speculative check instructions (`chk.s`).

Table 4-82. Unit Masks for INST_FAILED_CHKS_RETIRED

Extension	PMC.umask [19:16]	Description
---	bxx00	(* nothing will be counted *)
INT	bxx01	only integer instructions
FP	bxx10	only floating-point instructions
ALL	bxx11	both integer and floating-point instructions

ISB_BUNPAIRS_IN

- **Title:** Bundle Pairs Written from L2I into FE
- **Category:** L1 Instruction Cache and prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x46, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Provides information about the number of bundle pairs (32 bytes) written from L2I (and beyond) into the front end.
- **NOTE:** This event is qualified with IBRP0 if the cache line was tagged as a demand fetch and IBRP1 if the cache line was tagged as a prefetch match.

ITLB_MISSES_FETCH

- **Title:** Instruction Translation Buffer Misses Demand Fetch
- **Category:** TLB **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x47, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of ITLB misses for demand fetch.

Table 4-83. Unit Masks for ITLB_MISSES_FETCH

Extension	PMC.umask [19:16]	Description
---	bxx00	(* nothing will be counted *)
L1ITLB	bxx01	All misses in L1ITLB will be counted. even if L1ITLB is not updated for an access (Uncacheable/nat page/not present page/faulting/some flushed), it will be counted here.
L2ITLB	bxx10	All misses in L1ITLB which also missed in L2ITLB will be counted.
ALL	bxx11	All tlb misses will be counted. Note that this is not equal to sum of the L1ITLB and L2ITLB umasks because any access could be a miss in L1ITLB and L2ITLB.

L1DTLB_TRANSFER

- **Title:** L1DTLB Misses that Hit in the L2DTLB for Accesses Counted in L1D_READS
- **Category:** TLB/L1D Cache Set 0 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xc0, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of times an L1DTLB miss hits in the L2DTLB for an access counted in L1D_READS.
- **NOTE:** This is a restricted set 0 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5. In code sequence a;;b if "a" takes an exception and "b" requires an L2DTLB->L1DTLB transfer, the transfer is performed but not counted in this event. This is necessary to remain consistent with L1D_READS which will not count "b" because it is not reached. If thread switch occurs in the middle of DET stall while a transfer is pending, this event won't be counted.

L1D_READS_SET0

- **Title:** L1 Data Cache Reads (Set 0)
- **Category:** L1 Data Cache/L1D Cache Set 0 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xc2, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of data memory read references issued into memory pipeline which are serviced by L1D (only integer loads), RSE loads, L1-hinted loads (L1D returns data if it hits in L1D but does not do a fill) and check loads (`ld.c`). Uncacheable reads, VHPT loads, semaphores, floating-point loads, and `lfetch` instructions are not counted here because L1D does not handle these. The count includes wrong path operations but excludes predicated off operations.
- **NOTE:** This is a restricted set 0 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5. Only ports 0 and 1 are measured.

L1D_READS_SET1

- **Title:** L1 Data Cache Reads (Set 1)
- **Category:** L1 Data Cache/L1D Cache Set 1 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xc4, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of data memory read references issued into memory pipeline which are serviced by L1D (only integer loads), RSE loads, L1-hinted loads (L1D returns data if it hits in L1D but does not do a fill) and check loads (`ld.c`). Uncacheable reads, VHPT loads, semaphores, floating-point loads and `lfetch` instructions are not counted here because L1D does not handle these. The count includes wrong path operations but excludes predicated off operations.
- **NOTE:** This is a restricted set 1 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5. Only ports 0 and 1 are measured.

L1D_READ_MISSES

- **Title:** L1 Data Cache Read Misses
- **Category:** L1 Data Cache/L1D Cache Set 1 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xc7, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of L1 Data Cache read misses. L1 Data Cache is write through; therefore write misses are not counted. The count only includes misses caused by references counted by L1D_READS event. It will include L1D misses which missed the ALAT but not

those which hit in the ALAT. Semaphores are not handled by L1D and are not included in this count

- **NOTE:** This is a restricted set 1 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5. Only ports 0 and 1 are measured.

Table 4-84. Unit Masks for L1D_READ_MISSES

Extension	PMC.umask [19:16]	Description
ALL	bxxx0	all L1D read misses will be counted.
RSE_FILL	bxxx1	only L1D read misses caused by RSE fills will be counted

L1ITLB_INSERTS_HPW

- **Title:** L1ITLB Hardware Page Walker Inserts
- **Category:** TLB **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x48, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of L1ITLB inserts done by Hardware Page Walker.

L1I_EAR_EVENTS

- **Title:** Instruction EAR Events
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x43, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of L1 Instruction Cache or L1ITLB events captured by EAR.

L1I_FETCH_ISB_HIT

- **Title:** “Just-In-Time” Instruction Fetch Hitting In and Being Bypassed from ISB
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x66, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Provides information about an instruction fetch hitting in and being bypassed from the ISB (Instruction Streaming Buffer). It will not count “critical bypasses,” i.e. anytime the pipeline has to stall waiting for data to be delivered from L2I. It will count “just-in-time bypasses,” i.e. when instruction data is delivered by the L2I in time for the instructions to be consumed without stalling the front-end pipe.
- **NOTE:** Demand fetches which hit the ISB at the same time as they are being transferred to the Instruction Cache (1 cycles window) will not be counted because they have to be treated as cache hits for the purpose of branch prediction. This event is qualified with IBRP0 if the cache line was tagged as a demand fetch and IBRP1 if the cache line was tagged as a prefetch match.

L1I_FETCH_RAB_HIT

- **Title:** Instruction Fetch Hitting in RAB
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x65, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Provides Information about instruction fetch hitting in the RAB.
- **NOTE:** This event is qualified with IBRP0 if the cache line was tagged as a demand fetch and IBRP1 if the cache line was tagged as a prefetch match.

L1I_FILLS

- **Title:** L1 Instruction Cache Fills
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x41, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Provides information about the number of line fills from ISB to the L1 Instruction Cache (64-byte chunks).
- **NOTE:** This event is qualified with IBRP0 if the cache line was tagged as a demand fetch or IBRP1 if the cache line was tagged as a prefetch match. It is impossible for this event to fire if the corresponding entry is not in L1ITLB

L1I_PREFETCHES

- **Title:** L1 Instruction Prefetch Requests
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x44, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Provides information about the number of issued L1 cache line prefetch requests (64 bytes/line). The reported number includes streaming and non-streaming prefetches (hits and misses in L1 Instruction Cache are both included).
- **NOTE:** This event is qualified with IBRP1

L1I_PREFETCH_STALL

- **Title:** Prefetch Pipeline Stalls
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x67, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Provides Information on why the prefetch pipeline is stalled.

Table 4-85. Unit Masks for L1I_PREFETCH_STALL

Extension	PMC.umask [19:16]	Description
---	bxx00-bxx01	(* nothing will be counted *)
FLOW	bxx10	Asserted when the streaming prefetcher is working close to the instructijons being fetched for demand reads, and is not asserted when the streaming prefetcher is ranging way ahead of the demand reads.
ALL	bxx11	Number of clocks prefetch pipeline is stalled

L1I_PURGE

- **Title:** L1ITLB Purges Handled by L1I
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x4b, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Provides information on the number of L1ITLB purges handled by L1I. This event is caused by a purge instruction, global purge from the bus cluster, inserts into L2ITLB. It is not the same as column invalidates which are done on L1ITLB.

L1I_PVAB_OVERFLOW

- **Title:** PVAB Overflow
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x69, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Provides Information about the Prefetch Virtual Address Buffer overflowing.

L1I_RAB_ALMOST_FULL

- **Title:** Is RAB Almost Full?
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x64, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Provides Information about Read Address Buffer being almost full.

L1I_RAB_FULL

- **Title:** Is RAB Full?
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x60, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Provides Information about Read Address Buffer being full.

L1I_READS

- **Title:** L1 Instruction Cache Reads
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x40, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Provides information about the number of demand fetch reads (i.e. all accesses regardless of hit or miss) to the L1 Instruction Cache (32-byte chunks).
- **NOTE:** Demand fetches which have an L1ITLB miss, and L1I cache miss, and collide with a fill-recirculate to icache, will not be counted in this event even though they will be counted in L2I_DEMAND_READS.

L1I_SNOOP

- **Title:** Snoop Requests Handled by L1I
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0x4a, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Provides information on the number of snoop requests (64-byte granular) handled by L1I.
- **NOTE:** Each “fc” instruction will produce 1 snoop request to L1I after it goes out on the bus. If IFR snoop pipeline is busy when L1D sends the snoop to IFR, this event will count more than once for the same snoop. A victimized line will also produce a snoop. Some bus transactions also can cause L1I snoops.

L1I_STRM_PREFETCHES

- **Title:** L1 Instruction Cache Line Prefetch Requests
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x5f, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Provides Information about the number of L1I cache line prefetch requests (64 bytes/line) which go through prefetch pipeline (i.e. hit or miss in L1I cache is not factored in) in streaming mode only (initiated by `br.many`).
- **NOTE:** This event is qualified with IBRP1

L2DTLB_MISSES

- **Title:** L2DTLB Misses
- **Category:** TLB/L1D Cache Set 0 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xc1, **Max. Inc/Cyc:** 4, **MT Capture Type:** A
- **Definition:** Counts the number of L2DTLB misses (which is the same as references to HPW; DTLB_HIT=0) for demand requests.
- **NOTE:** This is a restricted set 0 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5. If HPW is enabled all the time, this event and HPW_DATA_REFERENCES are equivalent. This will include misses the L2DTLB did not squash even though the instructions causing the miss did not get to retirement. If thread switch occurs in the middle of DET stall while this is pending, this event won't be reported.

L2D_BAD_LINES_SELECTED

- **Title:** Valid Line Replaced When Invalid Line Is Available
- **Category:** L2 Data Cache Set 5 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xec, **Max. Inc/Cyc:** 4, **MT Capture Type:** F
- **Definition:** Counts the number of times a valid line was selected for replacement when an invalid line was available.
- **NOTE:** This is a restricted set 5 L2D Cache event that is paired with L2D_STORE_HIT_SHARED. Active thread is used as an approximation for this count.

Table 4-86. Unit Masks for L2D_BAD_LINES_SELECTED

Extension	PMC.umask [19:16]	Description
ANY	b0xxx	Valid line replaced when invalid line is available

L2D_BYPASS

- **Title:** Count L2D Bypasses
- **Category:** L2 Data Cache Set 1 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xe4, **Max. Inc/Cyc:** 4, **MT Capture Type:** F
- **Definition:** Counts the number of times a bypass occurred.
- **NOTE:** This is a restricted L2D cache event that is paired with L2D_OZQ_RELEASE. Active thread is the true thread for the count. The Itanium 2 processor version of this count was too speculative to be useful. On Montecito it now counts only bypasses that were successful. It also

supplies a count of the number of bypasses, rather than just an indication that a bypass occurred. Note that two of the bypass counts are not .all capable.

Table 4-87. Unit Masks for L2D_BYPASS

Extension	PMC.umask [19:16]	Description
L2_DATA1	bxx00	Count only L2D hit data bypasses (L1D to L2A). **NOTE: Not .all capable.**
L2_DATA2	bxx01	Count only L2D hit data bypasses (L1W to L2I). **NOTE: Not .all capable.**
L3_DATA1	bxx10	Count only L3 data bypasses (L1D to L2A). **NOTE: Is .all capable.**
---	bxx11	(* nothing will be counted *)

L2D_FILLB_FULL

- **Title:** L2D Fill Buffer Is Full
- **Category:** L2 Data Cache Set 7 **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xf1, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of times L2D Fill Buffer is full.
- **NOTE:** This is a restricted set 7 L2D Cache event that is paired with L2D_OPS_ISSUED. The active thread is used as an approximation for this count. This event is not .all capable.

Table 4-88. Unit Masks for L2D_FILLB_FULL

Extension	PMC.umask [19:16]	Description
THIS	b0000	L2D Fill buffer is full. **NOTE: Not .all capable.**
---	b0001-b1111	(* count is undefined *)

L2D_FILL_MESI_STATE

- **Title:** L2D Cache Fills with MESI state
- **Category:** L2 Data Cache Set 8 **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xf2, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of times the L2D cache is filled with a particular MESI state.
- **NOTE:** This is a restricted set 8 L2D Cache event that is paired with L2D_VICTIMB_FULL. This event uses the true thread id for the filling operation. Note that the fill states of I and P cor-

respond to inflight snoop vs. fill conflicts. The addition of the I and P counts would equal the count of L2_SYNTH_PROBE count from the Itanium 2 processor.

Table 4-89. Unit Masks for L2D_FILL_MESI_STATE

Extension	PMC.umask [19:16]	Description
M	bx000	Modified
E	bx001	Exclusive
S	bx010	Shared
I	bx011	Invalid
P	bx1xx	Pending

L2D_FORCE_RECIRC

- **Title:** Forced Recirculates
- **Category:** L2 Data Cache Set 4 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xea, **Max. Inc/Cyc:** 4, **MT Capture Type:** F
- **Definition:** Counts the number of L2D ops forced to recirculate on insertion to OZQ, with the exception of SNP_OR_L3. SNP_OR_L3 will measure the number of times L2D ops are forced to recirculate. Anywhere from 0-32 ops can be affected by this one. All categories with the exception of TRAN_PERF, and SNP_OR_L3 occur at the insertion into the OZQ. SNP_OR_L3 is when an existing OZQ entry is forced to recirculate because an incoming snoop request matched its address or an access is issued to the L3/BC which will fill the same way/index this OZQ_ENTRY has “hit” in. TRAN_PREF is when an existing OZQ access is transformed into a prefetch. This event has changed significantly since the Itanium 2 processor due to the complexity of trying to use these counts to generate L2D cache hit and miss rates. The L0W,OZQ_MISS,and FILL_HIT counts have been OR’d together to create the SECONDARY counts, which include separate qualifications on reads or writes (semaphores and read modify write stores will be counted on both). This allows secondary misses (the term for an operation that inserts into OZQ as a miss when a miss for the same line is already outstanding) to be counted from only one set event. The LIMBO count was also added to see how many operations were inserting into limbo on there initial insert. The VIC_PEND count has been removed to make room since it usually only implied an L0W event.
- **NOTE:** This is a restricted set 4 L2D Cache event that is paired with L2D_ISSUED_RECIRC_OZQ_ACC. Active thread is the correct thread or an approximation depending on the umask. Some umasks are not .all capable.

Table 4-90. Unit Masks for L2D_FORCE_RECIRC (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
RECIRC	b00x0	Counts inserts into OzQ due to a recirculate. The recirculate due to secondary misses or various other conflicts **NOTE: Active Thread is Accurate. Is .all capable.**
LIMBO	b00x1	Count operations that went into the LIMBO Ozq state. This state is entered when the the op sees a FILL_HIT or OZQ_MISS event. **NOTE: Active Thread is Accurate. Is .all capable.**

Table 4-90. Unit Masks for L2D_FORCE_RECIRC (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
TAG_NOTOK	b0100	Count only those caused by L2D hits caused by in flight snoops, stores with a sibling miss to the same index, sibling probe to the same line or a pending mf.a instruction. This count can usually be ignored since its events are rare, unpredictable, and/or show up in one of the other events. **NOTE: Active Thread is Accurate. Not .all capable.**
TRAN_PREF	b0101	Count only those caused by L2D miss requests that transformed to prefetches **NOTE: Active Thread is Approximation. Not .all capable.**
SNP_OR_L3	b0110	Count only those caused by a snoop or L3 issue. **NOTE: Active Thread is Approximation. Not .all capable.**
TAG_OK	b0111	Count operations that inserted to Ozq as a hit. Thus it was NOT forced to recirculate. Likely identical to L2D_INSERT_HITS. **NOTE: Active Thread is Accurate. Not .all capable.**
FILL_HIT	b1000	Count only those caused by an L2D miss which hit in the fill buffer. **NOTE: Active Thread is Accurate. Is .all capable.**
FRC_RECIRC	b1001	Caused by an L2D miss when a force recirculate already existed in the Ozq. **NOTE: Active Thread is Accurate. Is .all capable.**
SAME_INDEX	b1010	Caused by an L2D miss when a miss to the same index was in the same issue group. **NOTE: Active Thread is Accurate. Is .all capable.**
OZQ_MISS	b1011	Caused by an L2D miss when an L2D miss was already in the OZQ. **NOTE: Active Thread is Accurate. Is .all capable.**
L1W	b1100	Count only those caused by a L2D miss one cycle ahead of the current op. **NOTE: Active Thread is Accurate. Is .all capable.**
SECONDARY_READ	b1101	Caused by L2D read op that saw a miss to the same address in OZQ, L2 fill buffer, or one cycle ahead in the main pipeline. **NOTE: Active Thread is Accurate. Is .all capable.**
SECONDARY_WRITE	b1110	Caused by L2D write op that saw a miss to the same address in OZQ, L2 fill buffer, or one cycle ahead in the main pipeline. **NOTE: Active Thread is Accurate. Is .all capable.**
SECONDARY_ALL	b1111	Caused by any L2D op that saw a miss to the same address in OZQ, L2 fill buffer, or one cycle ahead in the main pipeline. **NOTE: Active Thread is Accurate. Is .all capable.**

L2D_INSERT_HITS

- **Title:** Count Number of Times an Inserting Data Request Hit in the L2D.
- **Category:** L2 Data Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb1, **Max. Inc/Cyc:** 4, **MT Capture Type:** F
- **Definition:** Counts the number of times a cacheable data request hit the L2D cache on its first lookup. Thus this event will not count secondary misses that eventually became hits, thus allowing a more reliable hit and miss rate calculation. This count is new for Montecito.
- **NOTE:** This is a NOT a restricted L2D Cache event. This event uses active thread for a true thread indication.

L2D_INSERT_MISSES

- **Title:** Count Number of Times an Inserting Data Request Missed the L2D.
- **Category:** L2 Data Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb0, **Max. Inc/Cyc:** 4, **MT Capture Type:** F
- **Definition:** Counts the number of times a cacheable data request missed the L2D cache on its first lookup. Thus this event will count secondary misses and other misses that were forced to recirculate. This allows for a more reliable miss rate calculation as compared to the method the Itanium 2 uses for counting L2_MISSES and adding a combination of L2_FORCE_RECIRC events.
- **NOTE:** This is a NOT a restricted L2D Cache event. This event uses active thread for a true thread indication.

L2D_ISSUED_RECIRC_OZQ_ACC

- **Title:** Count Number of Times a Recirculate Issue Was Attempted and Not Preempted
- **Category:** L2 Data Cache Set 4 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xeb, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of times a recirculate was attempted that didn't get preempted by a fill/confirm/evervalid (fill/confirm tag updates have higher priority) or by an older sibling issuing a recirculate (only one recirculate can be sent per clock). This value can be added to L2D_OZQ_CANCELS*.RECIRC for the total number of times the L2D issue logic attempted to issue a recirculate.
- **NOTE:** This is a restricted L2D Cache event that is paired with L2D_FORCE_RECIRCULATE. This event uses the true thread id of the given recirculate.

L2D_L3ACCESS_CANCEL

- **Title:** L2D Access Cancelled by L2D
- **Category:** L2 Data Cache Set 3 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xe8, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of canceled L3 accesses. A unit mask, as specified in the following table, narrows this event down to a specific reason for the cancel. This event includes one event for dynamic throttling of L2D transform to prefetches and L2D Ozq tail collapse disabling.
- **NOTE:** This is a restricted set 3 L2D Cache event that is paired with L2D_OZDB_FULL. The L2D reject events used their true thread id. The dynamic disabling and non-coverage events use active thread as an approximation. None of the umasks in this event are .all capable. umasks ‘bx000 to ‘bx011 use the true tread of the access. umasks ‘bx100 to ‘bx111 use the active thread.

Table 4-91. Unit Masks for L2D_L3ACCESS_CANCEL

Extension	PMC.umask [19:16]	Description
INV_L3_BYP	bx000	L2D cancelled a bypass because it did not commit, or was not a valid opcode to bypass, or was not a true miss of L2D (either hit,recirc,or limbo) **NOTE: Active thread is accurate.**
SPEC_L3_BYP	bx001	L2D cancelled speculative L3 bypasses because it was not a WB memory attribute or it was an effective release. **NOTE: Active thread is accurate.**
ANY	bx010	count cancels due to any reason. This umask will count more than the sum of all the other umasks. It will count things that weren't committed accesses when they reached L1w, but the L2D attempted to bypass them to the L3 anyway (speculatively). This will include accesses made repeatedly while the main pipeline is stalled and the L1d is attempting to recirculate an access down the L1d pipeline. Thus, an access could get counted many times before it really does get bypassed to the L3. It is a measure of how many times we asserted a request to the L3 but didn't confirm it. **NOTE: Active thread is accurate.**
ER_REJECT	bx011	Count only requests that were rejected by ER **NOTE: Active thread is accurate.**
P2_COV_SNP_TEM	bx100	A snoop saw an L2D tag error and missed **NOTE: Active thread is approximation.**
P2_COV_SNP_VIC	bx101	A snoop hit in the L1D victim buffer **NOTE: Active thread is approximation.**
P2_COV_SNP_FILL_NOSNP	bx110	A snoop and a fill to the same address reached the L2D within a 3 cycle window of each other or a snoop hit a nosnoops entry in Ozq. **NOTE: Active thread is approximation.**
TAIL_TRANS_DIS	bx111	Count the number of cycles that either transform to prefetches or Ozq tail collapse have been dynamically disabled. This would indicate that memory contention has lead the L2D to throttle request to prevent livelock scenarios. **NOTE: Active thread is approximation.**

L2D_MISSES

- **Title:** L2D Misses
- **Category:** L2 Data Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xcb, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of L2D cache misses (in terms of the number of L2D cache line requests sent to L3). It includes all cacheable data requests. This event does not count secondary L2D misses. To count secondary misses as well as primary misses use the new L2D_INSERT_MISSES count.
- **NOTE:** This count is not set restricted. This count uses its true thread id.

L2D_OPS_ISSUED

- **Title:** Operations Issued By L2D
- **Category:** L2 Data Cache Set 7 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xf0, **Max. Inc/Cyc:** 4, **MT Capture Type:** F
- **Definition:** Counts the number of operations issued by L2D as hits as specified by the operation type. This does not count operations that were cancelled. This event will count operations that originally missed L2D but then became hits and completed.
- **NOTE:** This is a restricted set 7 L2D Cache event that is paired with L2D_FILLB_FULL. This count uses the true thread id. This event incorrectly counted semaphores under OTHER in the Itanium 2 processor. Montecito has fixed that bug and added an lfetch count as well. None of these events are .all capable.

Table 4-92. Unit Masks for L2D_OPS_ISSUED

Extension	PMC.umask [19:16]	Description
INT_LOAD	bx000	Count only valid integer loads, including ld16.
FP_LOAD	bx001	Count only valid floating-point loads
RMW	bx010	Count only valid read_modify_write stores and semaphores including cmp8xchg16.
STORE	bx011	Count only valid non-read_modify_write stores, including st16.
LFETCH	bx1x0	Count only lfetch operations.
OTHER	bx1x1	Count only valid non-load, no-store accesses that are not listed above.

L2D_OZDB_FULL

- **Title:** L2D OZ Data Buffer Is Full
- **Category:** L2 Data Cache Set 3 **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xe9, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of cycles the L2D Oz Data Buffer is full.
- **NOTE:** This is a restricted set L2D Cache event that is paired with L2D_L3_ACCESS_CANCELS. This event uses the active thread id for an approximation.

Table 4-93. Unit Masks for L2D_OZDB_FULL

Extension	PMC.umask [19:16]	Description
THIS	b0000	L2D OZ Data Buffer is full
---	b0001-b1111	(* count is undefined *)

L2D_OZQ_ACQUIRE

- **Title:** Acquire Ordering Attribute Exists in L2D OZQ
- **Category:** L2 Data Cache Set 6 **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xef , **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of clocks for which the “acquire” ordering attribute existed in the L2D OZ Queue.
- **NOTE:** This is a restricted set 6 L2D Cache event. This event uses active thread as an approximation.

L2D_OZQ_CANCELS0

- **Title:** L2D OZQ Cancels (Specific Reason Set 0)
- **Category:** L2 Data Cache Set 0 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xe0, **Max. Inc/Cyc:** 4, **MT Capture Type:** F
- **Definition:** Counts the number of total L2D OZ Queue Cancels due to a specific reason (based on umask).
- **NOTE:** This is a restricted set 0 L2D Cache event that is grouped with L2D_OZQ_CANCELS1 and L2D_OZQ_FULL. Only 1 of the 2 L2D_OZQ_CANCEL events may be measured at any given time. This event counts with the true thread id of the operation. Compared to the Itanium 2 processor, several counts have been removed due to their rare occurrence, or due to ifetch removal. They are ECC, SCRUB, D_IFETCH, and HPW_IFETCH_CONF. Other related events were combined together to reduce the number of sample intervals required to collect all of the umasked events. Those that were combined are mentioned in the description column.

Table 4-94. Unit Masks for L2D_OZQ_CANCELS0 (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
RECIRC	b0000	a recirculate was cancelled due h/w limitations on recirculate issue rate. This is the combination of following subevents that were available separately in the Itanium® 2 processor: RECIRC_OVER_SUB: (caused by a recirculate oversubscription) DIDNT_RECIRC: (caused because it did not recirculate) WEIRD: (counts the cancels caused by attempted 5-cycle bypasses for non-aligned accesses and bypasses blocking recirculates for too long)
CANC_L2M_TO_L2C_ST	b0001	caused by a canceled store in L2M,L2D or L2C. This is the combination of following subevents that were available separately in the Itanium 2 processor: CANC_L2M_ST: (caused by canceled store in L2M) CANC_L2D_ST: (caused by canceled store in L2D) CANC_L2C_ST: (caused by canceled store in L2C)
L2A_ST_MAT	b0010	canceled due to an uncanceled store match in L2A
L2M_ST_MAT	b0011	canceled due to an uncanceled store match in L2M
L2D_ST_MAT	b0100	canceled due to an uncanceled store match in L2D
L2C_ST_MAT	b0101	canceled due to an uncanceled store match in L2C
ACQ	b0110	caused by an acquire somewhere in Ozq or ER.
REL	b0111	a release was cancelled due to some other operation

Table 4-94. Unit Masks for L2D_OZQ_CANCELS0 (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
BANK_CONF	b1000	a bypassed L2D hit operation had a bank conflict with an older sibling bypass or an older operation in the L2D pipeline.
SEMA	b1001	a semaphore op was cancelled for various ordering or h/w restriction reasons. This is the combination of following subevents that were available separately in the Itanium® 2 processor: SEM: (a semaphore) CCV: (a CCV)
OVER_SUB	b1010	a high Ozq issue rate resulted in the L2D having to cancel due to hardware restrictions. This is the combination of following subevents that were available separately in the Itanium 2 processor: OVER_SUB: (oversubscription) L1DF_L2M: (L1D fill in L2M)
OZQ_PREEMPT	b1011	an L2D fill return conflicted with, and cancelled, an ozq request for various reasons. Formerly known as L1_FILL_CONF.
WB_CONF	b1100	an OZQ request conflicted with an L2D data array read for a writeback. This is the combination of following subevents that were available separately in the Itanium 2 processor: READ_WB_CONF: (a write back conflict) ST_FILL_CONF: (a store fill conflict)
MISC_ORDER	b1101	a sync.i or mf.a . This is the combination of following subevents that were available separately in the Itanium 2 processor: SYNC: (caused by sync.i) MFA: (a memory fence instruction)
FILL_ST_CONF	b1110	an OZQ store conflicted with a returning L2D fill
OZDATA_CONF	b1111	an OZQ operation that needed to read the OZQ data buffer conflicted with a fill return that needed to do the same.

L2D_OZQ_CANCELS1

- **Title:** L2D OZQ Cancels (Late or Any)
- **Category:** L2 Data Cache Set 0 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xe2, **Max. Inc/Cyc:** 4, **MT Capture Type:** F
- **Definition:** Counts the number of total L2D OZ Queue Cancels (regardless of reason) or L2D OZ Queue Cancels due to a specific reason (based on umask).
- **NOTE:** This is a restricted set 0 L2D Cache event that is grouped with L2D_OZQ_CANCELS1 and L2D_OZQ_FULL. Only 1 of the 2 L2D_OZQ_CANCEL events may be measured at any given time. This event counts with the true thread id of the operation

Table 4-95. Unit Masks for L2D_OZQ_CANCELS0

Extension	PMC.umask [19:16]	Description
ANY	bxx00	counts the total OZ Queue cancels
LATE_SPEC_BYP	bxx01	counts the late cancels caused by speculative bypasses
SIBLING_ACQ_REL	bxx10	counts the late cancels caused by releases and acquires in the same issue group. This is the combination of following subevents that were available separately in the Itanium® 2 processor: LATE_ACQUIRE: (late cancels caused by acquires) LATE_RELEASE: (late cancels caused by releases)
LATE_BYP_EFFRELEASE	bxx11	counts the late cancels caused by L1D to L2A bypass effective releases

L2D_OZQ_FULL

- **Title:** L2D OZQ Is Full
- **Category:** L2 Data Cache Set 0 **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xe1, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of times L2D OZQ is full.
- **NOTE:** This is a restricted set 0 L2D cache event that is grouped with L2D_OZQ_CANCELS0/1. This event uses active thread as an approximation. This event is not .all capable.

Table 4-96. Unit Masks for L2D_OZQ_FULL

Extension	PMC.umask [19:16]	Description
THIS	b0000	L2D OZQ is full. **NOTE: Not .all capable.**
---	b0001-b1111	(* count is undefined *)

L2D_OZQ_RELEASE

- **Title:** Release Ordering Attribute Exists in L2D OZQ
- **Category:** L2 Data Cache Set 1 **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xe5, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of clocks entries with an “effective release” ordering attribute existed in the L2D OZ Queue. This includes not just architected .rel instructions, but also effective releases due to loads and stores to the same 4 byte chunk.
- **NOTE:** This is a restricted set 1L2D Cache event that is paired with L2D_BYPASS. This event uses active thread as an approximation. This event is not .all capable.

L2D_REFERENCES

- **Title:** Data Read/Write Access to L2D
- **Category:** L2 Data Cache Set 2 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xe6, **Max. Inc/Cyc:** 4, **MT Capture Type:** F
- **Definition:** Counts the number of requests made to L2D due to a data read and/or write access. Semaphore operations are counted as one read and one write.
- **NOTE:** This is a restricted set 2 L2D cache event that does not share an event code. Active thread is the true thread for the count.

Table 4-97. Unit Masks for L2D_REFERENCES

Extension	PMC.umask [19:16]	Description
---	bxx00	(* nothing will be counted *)
READS	bxx01	count only data read and semaphore operations.
WRITES	bxx10	count only data write and semaphore operations
ALL	bxx11	count both read and write operations (semaphores will count as 2)

L2D_STORE_HIT_SHARED

- **Title:** Store Hit a Shared Line
- **Category:** L2 Data Cache Set 5 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xed, **Max. Inc/Cyc:** 2, **MT Capture Type:** F
- **Definition:** Counts the number of times a store hit a shared line.
- **NOTE:** This is a restricted set 5 L2D Cache event that is paired with L2D_NUM_BAD_LINES_SELECTED. This event uses active thread as an approximation.

Table 4-98. Unit Masks for L2D_STORE_HIT_SHARED

Extension	PMC.umask [19:16]	Description
ANY	b0xxx	Store hit a shared line

L2D_VICTIMB_FULL

- **Title:** L2D Victim Buffer Is Full
- **Category:** L2 Data Cache Set 8 **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xf3, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of times L2D Victim Buffer is full.
- **NOTE:** This is a restricted set 8 L2D Cache event that is paired with L2D_FILL_MESI_STATE. This event uses active thread for an approximation.

Table 4-99. Unit Masks for L2D_VICTIMB_FULL

Extension	PMC.umask [19:16]	Description
THIS	b0000	L2D victim buffer is full
---	b0001-b1111	(* count is undefined *)

L2I_DEMAND_READS

- **Title:** L2 Instruction Demand Fetch Requests
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x42, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of instruction requests to L2I due to L1I demand fetch misses. This event counts the number of demand fetches that miss both the L1I and the ISB regardless of whether they hit or miss in the RAB.
- **NOTE:** If a demand fetch does not have an L1ITLB miss, L2I_DEMAND_READS and L1I_READS line up in time. If a demand fetch does not have an L2ITLB miss, L2I_DEMAND_READS follows L1I_READS by 3-4 clocks (unless a flushed iwalk is pending ahead of it; which will increase the delay until the pending iwalk is finished). If demand fetch has an L2ITLB miss, the skew between L2I_DEMAND_READS and L1I_READS is not deterministic.

L2I_HIT_CONFLICTS

- **Title:** L2I hit conflicts
- **Category:** L2 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x7d, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** The number of times a tagged demand or prefetch request from IFR to L2I would have hit a valid line in the L2I cache, except that L2I was forced to abandon the lookup, either because the internal L2I read buffers don't have enough room left to buffer the read data (due to simultaneous higher-priority L3/BC fill data returns), or because a simultaneous snoop might be invalidating the line. In rare cases, this event may be signalled multiple times for a single request from IFR. The final time the request moves down the L2I pipeline, it also reports a single

L2I_READS.HIT.DMND or L2I_READS.MISS.DMND(PFTCH) (or L2I_UC_READS.DMND(PFTCH)) event, as appropriate.

- **NOTE:** This event is qualified with IBRP1

Table 4-100. Unit Masks for L2I_HIT_CONFLICTS

Extension	PMC.umask [19:16]	Description
---	00xx	None is counted
HIT.NONE	0100	None is counted
HIT.DMND	0101	Only demand fetches that hit the L2I are counted
HIT.PFTCH	0110	Only prefetches that hit the L2I are counted
HIT.ALL	0111	All fetches that hit the L2I are counted
MISS.NONE	1000	None is counted
MISS.DMND	1001	Only demand fetches that miss the L2I are counted
MISS.PFTCH	1010	Only prefetches that miss the L2I are counted
MISS.ALL	1011	All fetches that miss the L2I are counted
ALL.NONE	1100	None is counted
ALL.DMND	1101	Demand fetches that reference L2I are counted
ALL.PFTCH	1110	Prefetches that reference L2I are counted
ALL.ALL	1111	All fetches that reference L2I are counted

L2I_L3_REJECTS

- **Title:** L3 rejects
- **Category:** L2 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x7c, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** The number of times a tagged demand and prefetch requests from IFR to L2I misses in the L2I cache and attempts to issue to L3/BC, but has its L3/BC request rejected. This could be the result of a queue full condition in L3/BC, or it could be due to a previous L2I or L2D request to L3/BC for the same address. In any case, the L2I keeps trying to send its request to L3/BC until it finally gets accepted. Note that this event is signalled once for every time an L3/BC request is rejected and sent again, and so it can fire multiple times for a single request from IFR. The request also reports a single L2I_READS.MISS.DMND(PFTCH) or L2I_UC_READS.DMND(PFTCH) (as appropriate) event.
- **NOTE:** This event is qualified with IBRP1.

Table 4-101. Unit Masks for L2I_L3_REJECTS (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
---	00xx	None is counted
HIT.NONE	0100	None is counted
HIT.DMND	0101	Only demand fetches that hit the L2I are counted
HIT.PFTCH	0110	Only prefetches that hit the L2I are counted
HIT.ALL	0111	All fetches that hit the L2I are counted
MISS.NONE	1000	None is counted
MISS.DMND	1001	Only demand fetches that miss the L2I are counted

Table 4-101. Unit Masks for L2I_L3_REJECTS (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
MISS.PFTCH	1010	Only prefetches that miss the L2I are counted
MISS.ALL	1011	All fetches that miss the L2I are counted
ALL.NONE	1100	None is counted
ALL.DMND	1101	Demand fetches that reference L2I are counted
ALL.PFTCH	1110	Prefetches that reference L2I are counted
ALL.ALL	1111	All fetches that reference L2I are counted

L2I_PREFETCHES

- **Title:** L2 Instruction Prefetch Requests
- **Category:** L1 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x45, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts number of prefetch requests issued to the L2I cache. The reported number includes streaming and non-streaming prefetches.
- **NOTE:** This event is qualified with IBRP1.

L2I_READS

- **Title:** L2I Cacheable Reads
- **Category:** L2 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x78, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Provides number of cacheable code reads handled by L2I. Event L2I_READS.MISS.ALL counts only the primary misses. Secondary misses are counted (may be multiple times) in L2I_RECIRCULATES event and finally counted once as L2I_READS.HIT.ALL
- **NOTE:** This event is qualified with IBRP1.

Table 4-102. Unit Masks for L2I_READS (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
---	00xx	None is counted
HIT.NONE	0100	None is counted
HIT.DMND	0101	Only demand fetches that hit the L2I are counted
HIT.PFTCH	0110	Only prefetches that hit the L2I are counted
HIT.ALL	0111	All fetches that hit the L2I are counted
MISS.NONE	1000	None is counted
MISS.DMND	1001	Only demand fetches that miss the L2I are counted
MISS.PFTCH	1010	Only prefetches that miss the L2I are counted
MISS.ALL	1011	All fetches that miss the L2I are counted
ALL.NONE	1100	None is counted
ALL.DMND	1101	Demand fetches that reference L2I are counted

Table 4-102. Unit Masks for L2I_READS (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
ALL.PFTCH	1110	Prefetches that reference L2I are counted
ALL.ALL	1111	All fetches that reference L2I are counted

L2I_RECIRCULATES

- **Title:** L2I recirculates
- **Category:** L2 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x7b, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** The number of times a tagged (demand or prefetch) request from IFR to L2I matches a line marked pending in the L2I cache. This means that L2I has already sent another request for this same address (but via a different RABID) to L3/BC, and is still waiting for its data to return. Thus, this new request cannot yet return data to IFR, but a duplicate request to L3/BC would just be rejected, so the request must enter a loop of waiting for an event (snoop or fill) which might change the status of the pending line, and then reissuing through the L2I pipeline; the cycle repeats until the request no longer hits pending. Note that this event is signalled every time a demand request reissues, and so it might fire multiple times for a single request from IFR. The final time the request moves down the L2I pipeline, it also reports either a single L2I_READS.HIT.DMND(PFTCH) or L2I_READS.MISS.DMND (or L2I_UC_READS.DMND) event
- **NOTE:** This event is qualified with IBRP1

Table 4-103. Unit Masks for L2I_RECIRCULATES

Extension	PMC.umask [19:16]	Description
---	00xx	None is counted
HIT.NONE	0100	None is counted
HIT.DMND	0101	Only demand fetches that hit the L2I are counted
HIT.PFTCH	0110	Only prefetches that hit the L2I are counted
HIT.ALL	0111	All fetches that hit the L2I are counted
MISS.NONE	1000	None is counted
MISS.DMND	1001	Only demand fetches that miss the L2I are counted
MISS.PFTCH	1010	Only prefetches that miss the L2I are counted
MISS.ALL	1011	All fetches that miss the L2I are counted
ALL.NONE	1100	None is counted
ALL.DMND	1101	Demand fetches that reference L2I are counted
ALL.PFTCH	1110	Prefetches that reference L2I are counted
ALL.ALL	1111	All fetches that reference L2I are counted

L2I_SPEC_ABORTS

- **Title:** L2I speculative aborts
- **Category:** L2 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x7e, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Provides number of speculative aborts. Event counts the number of times a tagged change-priority-to-demand request from IFR to L2I starts down the L2I pipeline, but L2I finds that it has already been looked up, and aborts the new lookup. This can occur because, in order to satisfy demand requests as quickly as possible, L1I starts to look up a demand request without waiting to check if that request has already been handled (as is possible if it was originally issued as a prefetch, and later the IFR changes its priority to a demand)
- **NOTE:** This event is qualified with IBRP1

L2I_SNOOP_HITS

- **Title:** L2I snoop hits
- **Category:** L2 Instruction Cache and Prefetch **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x7f, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** The number of times a snoop request to L2I hits a valid line in the L2I cache and must invalidate that line. The thread ID reported for this event is simply the currently active thread.
- **NOTE:** This event is NOT qualified with tags

L2I_UC_READS

- **Title:** L2I uncacheable reads
- **Category:** L2 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x79, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Provides number of UC code reads handled by L2I.
- **NOTE:** This event is qualified with IBRP1.

Table 4-104. Unit Masks for L2I_UC_READS (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
---	00xx	None is counted
HIT.NONE	0100	None is counted
HIT.DMND	0101	Only demand fetches that hit the L2I are counted
HIT.PFTCH	0110	Only prefetches that hit the L2I are counted
HIT.ALL	0111	All fetches that hit the L2I are counted
MISS.NONE	1000	None is counted
MISS.DMND	1001	Only demand fetches that miss the L2I are counted
MISS.PFTCH	1010	Only prefetches that miss the L2I are counted
MISS.ALL	1011	All fetches that miss the L2I are counted
ALL.NONE	1100	None is counted
ALL.DMND	1101	Demand fetches that reference L2I are counted

Table 4-104. Unit Masks for L2I_UC_READS (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
ALL.PFTCH	1110	Prefetches that reference L2I are counted
ALL.ALL	1111	All fetches that reference L2I are counted

L2I_VICTIMIZATION

- **Title:** L2I victimizations
- **Category:** L2 Instruction Cache and Prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x7a, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** The number of times a tagged demand or prefetch request from IFR to L2I misses in the L2I cache, and needs to replace a valid line. The request also reports a single L2I_READS.MISS.DMND event.
- **NOTE:** This event is qualified with IBRP1

L3_INSERTS

- **Title:** L3 Cache Lines inserts
- **Category:** L3 Unified Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xda, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of valid L3 lines that have been written to the L3 due to a L2 miss allocation.
- **NOTE:** This event may be qualified by MESI. If this event is filtered by the PMC's MESI bits, the filter will apply to the current cache line rather than the incoming line. To measure **all** events, the MESI filter must be set to b1111.

L3_LINES_REPLACED

- **Title:** L3 Cache Lines Replaced
- **Category:** L3 Unified Cache **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xdf, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of valid L3 lines (dirty victims) that have been replaced. Exclusive clean/shared and clean castouts may also be counted depending on platform specific settings.
- **NOTE:** This event may be qualified by MESI. If this event is filtered by the PMC's MESI bits, the filter will apply to the current cache line rather than the incoming line.To measure **all** events, the MESI filter must be set to b1111.

L3_MISSES

- **Title:** L3 Misses
- **Category:** L3 Unified Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xdc, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of L3 cache misses. Includes misses caused by instruction fetch, data read/write, L2D write backs and the HPW.

L3_READS

- **Title:** L3 Reads
- **Category:** L3 Unified Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xdd, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of L3 cache read accesses.
- **NOTE:** This event may be qualified by MESI.To measure **all** events, the MESI filter must be set to b1111.

Table 4-105. Unit Masks for L3_READS

Extension	PMC.umask [19:16]	Description
---	b0000	(* nothing will be counted *)
DINST_FETCH.HIT	b0001	L3 Demand Instruction Fetch Hits
DINST_FETCH.MISS	b0010	L3 Demand Instruction Fetch Misses
DINST_FETCH.ALL	b0011	L3 Demand Instruction References
---	b0100	(* nothing will be counted *)
INST_FETCH.HIT	b0101	L3 Instruction Fetch and Prefetch Hits
INST_FETCH.MISS	b0110	L3 Instruction Fetch and Prefetch Misses
INST_FETCH.ALL	b0111	L3 Instruction Fetch and Prefetch References
---	b1000	(* nothing will be counted *)
DATA_READ.HIT	b1001	L3 Load Hits (excludes reads for ownership used to satisfy stores)
DATA_READ.MISS	b1010	L3 Load Misses (excludes reads for ownership used to satisfy stores)
DATA_READ.ALL	b1011	L3 Load References (excludes reads for ownership used to satisfy stores)
---	b1100	(* nothing will be counted *)
ALL.HIT	b1101	L3 Read Hits
ALL.MISS	b1110	L3 Read Misses
ALL.ALL	b1111	L3 Read References

L3_REFERENCES

- **Title:** L3 References
- **Category:** L3 Unified Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xdb, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of L3 accesses. Includes instruction fetch/prefetch, data read/write and L2D write backs.

L3_WRITES

- **Title:** L3 Writes
- **Category:** L3 Unified Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xde, **Max. Inc/Cyc:** 1, **MT Capture Type:** F
- **Definition:** Counts the number of L3 cache write accesses.
- **NOTE:** This event may be qualified by MESI. If this event is filtered by the PMC's MESI bits, the filter will apply to the current cache line rather than the incoming line.To measure **all** events, the MESI filter must be set to b1111.

Table 4-106. Unit Masks for L3_WRITES

Extension	PMC.umask [19:16]	Description
---	b00xx	(* nothing will be counted *)
---	b0100	(* nothing will be counted *)
DATA_WRITE.HIT	b0101	L3 Store Hits (excludes L2D write backs, includes L3 read for ownership requests that satisfy stores)
DATA_WRITE.MISS	b0110	L3 Store Misses (excludes L2D write backs, includes L3 read for ownership requests that satisfy stores)
DATA_WRITE.ALL	b0111	L3 Store References (excludes L2D write backs, includes L3 read for ownership requests that satisfy stores)
---	b1000	(* nothing will be counted *)
L2_WB.HIT	b1001	L2D Write Back Hits
L2_WB.MISS	b1010	L2D Write Back Misses
L2_WB.ALL	b1011	L2DWrite Back References
---	b1100	(* nothing will be counted *)
ALL.HIT	b1101	L3 Write Hits
ALL.MISS	b1110	L3 Write Misses
ALL.ALL	b1111	L3 Write References

LOADS_RETIRED

- **Title:** Retired Loads
- **Category:** Instruction Execution/L1D Cache Set 3 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xcd, **Max. Inc/Cyc:** 4, **MT Capture Type:** A
- **Definition:** Counts the number of retired loads, excluding predicated off loads. The count includes integer, floating-point, RSE, semaphores, VHPT, uncacheable loads and check loads (`ld.c`) which missed in ALAT and L1D (because this is the only time this looks like any other load). Also included are loads generated by squashed HPW walks.
- **NOTE:** This is a restricted set 3 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5.

LOADS_RETIRED_INTG

- **Title:** Integer loads retired
- **Category:** Instruction Execution/L1D Cache Set 3 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xd8, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of retired integer loads.
- **NOTE:** This is a restricted set 6 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5.

MEM_READ_CURRENT

- **Title:** Current Mem Read Transactions On Bus
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x89, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Counts the number of current memory read transactions (BRC) on the bus.

Table 4-107. Unit Masks for MEM_READ_CURRENT

Extension	PMC.umask [19:16]	Description
---	bxx00	(* illegal selection *)
IO	bxx01	non-CPU priority agents
---	bxx10	(* illegal selection *)
ANY	bxx11	CPU or non-CPU (all transactions).

MISALIGNED_LOADS_RETIRED

- **Title:** Retired Misaligned Load Instructions
- **Category:** Instruction Execution/L1D Cache Set 3 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xce, **Max. Inc/Cyc:** 4, **MT Capture Type:** A
- **Definition:** Counts the number of retired misaligned load instructions, excluding those that were predicated off. It includes integer, floating-point loads, semaphores and check loads (`ld.c`) which missed in ALAT and L1D (the only time this looks like any other load).
- **NOTE:** If a misaligned load takes a trap then it will not be counted here since only retired loads are counted. `PSR.ac` = 0 and not crossing the 0-7 or 8-15 byte boundary is the only time it will not trap. This is a restricted set 3 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5.

MISALIGNED_STORES_RETIRED

- **Title:** Retired Misaligned Store Instructions
- **Category:** Instruction Execution/L1D Cache Set 4 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xd2, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of retired misaligned store instructions, excluding those that were predicated off. It includes integer, floating-point, semaphores and uncacheable stores. Predicated off operations are not counted.
- **NOTE:** If a misaligned store takes a trap then it will not be counted here since only retired stores are counted. `PSR.ac` = 0 and not crossing the 0-15 byte boundary of a WB page is the

only time it will not trap. This is a restricted set 4 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5. Only ports 2 and 3 are counted.

NOPS_RETIRED

- **Title:** Retired NOP Instructions
- **Category:** Instruction Execution **IAR/DAR/OPC:** YN/Y
- **Event Code:** 0x50, **Max. Inc/Cyc:** 6, **MT Capture Type:** A
- **Definition:** Provides information on number of retired `nop.i`, `nop.m`, and `nop.b`,`nop.f` instructions, excluding `nop` instructions that were predicated off.

PREDICATE_SQUASHED_RETIRED

- **Title:** Instructions Squashed Due to Predicate Off
- **Category:** Instruction Execution **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x51, **Max. Inc/Cyc:** 6, **MT Capture Type:** A
- **Definition:** Provides information on number of instructions squashed due to a false qualifying predicate. Includes all non-B-syllable instructions which reached retirement with a false predicate.

RSE_CURRENT_REGS_2_TO_0

- **Title:** Current RSE Registers (Bits 2:0)
- **Category:** RSE Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x2b, **Max. Inc/Cyc:** 7, **MT Capture Type:** A
- **Definition:** Counts the number of current RSE registers before an RSE_EVENT_RETIRED occurred. The Montecito processor can have a total of 96 per cycle. The lowest 3 bits are stored in this counter (bits 2:0).

RSE_CURRENT_REGS_5_TO_3

- **Title:** Current RSE Registers (Bits 5:3)
- **Category:** RSE Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x2a, **Max. Inc/Cyc:** 7, **MT Capture Type:** A
- **Definition:** Counts the number of current RSE registers before an RSE_EVENT_RETIRED occurred. The Montecito processor can have a total of 96 per cycle. The middle 3 bits are stored in this counter (bits 5:3).

RSE_CURRENT_REGS_6

- **Title:** Current RSE Registers (Bit 6)
- **Category:** RSE Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x26, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of current RSE registers before an RSE_EVENT_RETIRED occurred. The Montecito processor can have a total of 96 per cycle. The highest 1 bit is stored in this counter (bit 6).

RSE_DIRTY_REGS_2_TO_0

- **Title:** Dirty RSE Registers (Bits 2:0)
- **Category:** RSE Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x29, **Max. Inc/Cyc:** 7, **MT Capture Type:** A
- **Definition:** Counts the number of dirty RSE registers before an RSE_EVENT_RETIRED occurred. The Montecito processor can have a total of 96 per cycle. The lowest 3 bits are stored in this counter (bits 2:0).

RSE_DIRTY_REGS_5_TO_3

- **Title:** Dirty RSE Registers (Bits 5:3)
- **Category:** RSE Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x28, **Max. Inc/Cyc:** 7, **MT Capture Type:** A
- **Definition:** Counts the number of dirty RSE registers before an RSE_EVENT_RETIRED occurred. The Montecito processor can have a total of 96 per cycle. The middle 3 bits are stored in this counter (bits 5:3).

RSE_DIRTY_REGS_6

- **Title:** Dirty RSE Registers (Bit 6)
- **Category:** RSE Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x24, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of dirty RSE registers before an RSE_EVENT_RETIRED occurred. The Montecito processor can have a total of 96 per cycle. The highest one bit is stored in this counter (bit 6).

RSE_EVENT_RETIRED

- **Title:** Retired RSE Operations
- **Category:** RSE Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x32, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of retired RSE operations (i.e. `alloc`, `br.ret`, `br.call`, loadrs, flushrs, cover, and rfi - see NOTE). This event is an indication of when instructions which affect the RSE are retired (which may or may not cause activity to memory subsystem).
- **NOTE:** The only time 2 RSE events can be retired in 1 clock are flushrs/call or flushrs/return bundles. These corner cases are counted as 1 event instead of 2 since this event is used to calculate the average number of current/dirty/invalid registers. rfi instructions will be included only if ifsvalid=1; which can be set either by using the cover instruction prior to the rfi, or explicitly setting the valid bit.

RSE_REFERENCES_RETIRED

- **Title:** RSE Accesses
- **Category:** RSE Events **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0x20, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of retired RSE loads and stores (Every time RSE.bof reaches RSE.storereg; otherwise known as mandatory events including rnat fills & spills). This event is an indication of when RSE causes activity to memory subsystem.
- **NOTE:** Privilege level for DBR tags is determined by the RSC register; but privilege level for IBR tags is determined by PSR.cpl. RSE traffic which is caused by rfi will be tagged by the target of the rfi.

Table 4-108. Unit Masks for RSE_REFERENCES_RETIRED

Extension	PMC.umask [19:16]	Description
---	bxx00	(* nothing will be counted *)
LOAD	bxx01	Only RSE loads will be counted.
STORE	bxx10	Only RSE stores will be counted.
ALL	bxx11	Both RSE loads and stores will be counted.

SERIALIZATION_EVENTS

- **Title:** Number of srlz.i Instructions
- **Category:** System Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x53, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of srlz.i instructions (because it causes a microtrap and an xpn-flush fires)

SI_CCQ_COLLISIONS

- **Title:** Clean Castout Queue Collisions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xa8, **Max. Inc/Cyc:** 2, **MT Capture Type:** C
- **Definition:** Counts the number of address collisions between a CCQ entry and incoming FSB transaction.

Table 4-109. Unit Masks for SI_CCQ_COLLISIONS

Extension	PMC.umask [19:16]	Description
EITHER	bxxx0	transactions initiated by either cpu core
SELF	bxxx1	transactions initiated by ‘this’ cpu core

SI_CCQ_INSERTS

- **Title:** Clean Castout Queue Insertions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xa5, **Max. Inc/Cyc:** 2, **MT Capture Type:** S
- **Definition:** Counts insertions into the CCQ.

Table 4-110. Unit Masks for SI_CCQ_INSERTS

Extension	PMC.umask [19:16]	Description
EITHER	bxxx0	transactions initiated by either cpu core
SELF	bxxx1	transactions initiated by 'this' cpu core

SI_CCQ_LIVE_REQ_HI

- **Title:** Clean Castout Queue Requests (upper bit)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xa7, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Counts the number of live CCQ requests. The Montecito processor can have a total of 8 per cycle. The upper bit is stored in this counter.

Table 4-111. Unit Masks for SI_CCQ_LIVE_REQ_HI

Extension	PMC.umask [19:16]	Description
EITHER	bxxx0	transactions initiated by either cpu core
SELF	bxxx1	transactions initiated by 'this' cpu core

SI_CCQ_LIVE_REQ_LO

- **Title:** Clean Castout Queue Requests (lower three bits)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xa6, **Max. Inc/Cyc:** 7, **MT Capture Type:** C
- **Definition:** Counts the number of live CCQ requests. The Montecito processor can have a total of 8 per cycle. The lower three bits are stored in this counter.

Table 4-112. Unit Masks for SI_CCQ_LIVE_REQ_LO

Extension	PMC.umask [19:16]	Description
EITHER	bxxx0	transactions initiated by either cpu core
SELF	bxxx1	transactions initiated by 'this' cpu core

SI_CYCLES

- **Title:** SI Cycles
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x8e, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Counts SI clock cycles. SI clock is: bus_ratio * bus_clock.

SI_IOQ_COLLISIONS

- **Title:** In Order Queue Collisions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xaa, **Max. Inc/Cyc:** 2, **MT Capture Type:** C
- **Definition:** Counts the number of address collisions between an IOQ entry and outgoing FSB transaction.

SI_IOQ_LIVE_REQ_HI

- **Title:** Inorder Bus Queue Requests (upper bit)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x98, **Max. Inc/Cyc:** 2, **MT Capture Type:** C
- **Definition:** Counts the number of live in-order bus requests. The Montecito processor can have a total of 8 per cycle. The upper bit is stored in this counter.

SI_IOQ_LIVE_REQ_LO

- **Title:** Inorder Bus Queue Requests (lower three bits)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x97, **Max. Inc/Cyc:** 3, **MT Capture Type:** C
- **Definition:** Counts the number of live in-order bus requests. The Montecito processor can have a total of 8 per cycle. The lower three bits are stored in this counter.

SI_RQ_INSERTS

- **Title:** Request Queue Insertions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x9e, **Max. Inc/Cyc:** 2, **MT Capture Type:** S
- **Definition:** Counts insertions into the RQ.

Table 4-113. Unit Masks for SI_RQ_INSERTS

Extension	PMC.umask [19:16]	Description
EITHER	bxxx0	transactions initiated by either cpu core
SELF	bxxx1	transactions initiated by ‘this’ cpu core

SI_RQ_LIVE_REQ_HI

- **Title:** Request Queue Requests (upper bit)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xa0, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Counts the number of live RQ requests. The Montecito processor can have a total of 8 per cycle. The upper bit is stored in this counter..

Table 4-114. Unit Masks for SI_RQ_LIVE_REQ_HI

Extension	PMC.umask [19:16]	Description
EITHER	bxxx0	transactions initiated by either cpu core
SELF	bxxx1	transactions initiated by 'this' cpu core

SI_RQ_LIVE_REQ_LO

- **Title:** Request Queue Requests (lower three bits)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x9f, **Max. Inc/Cyc:** 7, **MT Capture Type:** C
- **Definition:** Counts the number of live RQ requests. The Montecito processor can have a total of 8 per cycle. The lower three bits are stored in this counter.

Table 4-115. Unit Masks for SI_RQ_LIVE_REQ_LO

Extension	PMC.umask [19:16]	Description
EITHER	bxxx0	transactions initiated by either cpu core
SELF	bxxx1	transactions initiated by 'this' cpu core

SI_SCB_INSERTS

- **Title:** Snoop Coalescing Buffer Insertions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xab, **Max. Inc/Cyc:** 4, **MT Capture Type:** C
- **Definition:** Counts the number of insertions into the snoop coalescing buffer as specified with the following Unit Mask. This is the 'raw' coherency answer of each core before being logically combined (see SCB_SIGNOFFS below).

Table 4-116. Unit Masks for SI_SCB_INSERTS (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
MISS.EITHER	b00x0	count MISS snoop signoffs (plus backsnoop inserts) from either cpu core
MISS.SELF	b00x1	count MISS snoop signoffs (plut backsnoop inserts) from 'this' cpu core
HIT.EITHER	b01x0	count HIT snoop signoffs from either cpu core
HIT.SELF	b01x1	count HIT snoop signoffs from 'this' cpu core
HITM.EITHER	b10x0	count HITM snoop signoffs from either cpu core

Table 4-116. Unit Masks for SI_SCB_INSERTS (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
HITM.SELF	b10x1	count HITM snoop signoffs from 'this' cpu core
ALL.EITHER	b11x0	count all snoop signoffs (plus backsnoop inserts) from either cpu core
ALL.SELF	b11x1	count all snoop signoffs (plut backsnoop inserts) from 'this' cpu core

SI_SCB_LIVE_REQ_HI

- **Title:** Snoop Coalescing Buffer Requests (upper bit)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xad, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Counts the number of live SCB requests. The Montecito processor can have a total of 8 per cycle. The upper bit is stored in this counter.

Table 4-117. Unit Masks for SI_SCB_LIVE_REQ_HI

Extension	PMC.umask [19:16]	Description
EITHER	bxxx0	transactions initiated by either cpu core
SELF	bxxx1	transactions initiated by 'this' cpu core

SI_SCB_LIVE_REQ_LO

- **Title:** Snoop Coalescing Buffer Requests (lower three bits)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xac, **Max. Inc/Cyc:** 7, **MT Capture Type:** C
- **Definition:** Counts the number of live SCB requests. The Montecito processor can have a total of 8 per cycle. The lower three bits are stored in this counter.

Table 4-118. Unit Masks for SI_SCB_LIVE_REQ_LO

Extension	PMC.umask [19:16]	Description
EITHER	bxxx0	transactions initiated by either cpu core
SELF	bxxx1	transactions initiated by 'this' cpu core

SI_SCB_SIGNOFFS

- **Title:** Snoop Coalescing Buffer Coherency Signoffs
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xae, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Counts the number of snoop signoffs driven from the snoop coalescing buffer as specified with the following Unit Mask. This is the 'final' coherency answer driven to FSB determined once the coherency answer from each core is collected.

Table 4-119. Unit Masks for SI_SCB_SIGNOFFS

Extension	PMC.umask [19:16]	Description
MISS	b00xx	count MISS snoop signoffs
HIT	b01xx	count HIT snoop signoffs
HITM	b10xx	count HITM snoop signoffs
ALL	b11xx	count all snoop signoffs

SI_WAQ_COLLISIONS

- **Title:** Write Address Queue Collisions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xa4, **Max. Inc/Cyc:** 1, **MT Capture Type:** S
- **Definition:** Counts the number of address collisions between a WAQ entry and incoming FSB transaction.

Table 4-120. Unit Masks for SI_CCQ_COLLISIONS

Extension	PMC.umask [19:16]	Description
EITHER	bxxx0	transactions initiated by either cpu core
SELF	bxxx1	transactions initiated by 'this' cpu core

SI_WDQ_ECC_ERRORS

- **Title:** Write Data Queue ECC Errors
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xaf, **Max. Inc/Cyc:** 2, **MT Capture Type:** S
- **Definition:** Counts the number of ECC errors detected in the WDQ. Qualified by the following Unit Mask.

Table 4-121. Unit Masks for SI_WDQ_ECC_ERRORS (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
SGL.EITHER	b00x0	count single-bit ecc errors from either cpu core
SGL.SELF	b00x1	count single-bit ecc errors from 'this' cpu core
DBL.EITHER	b01x0	count double-bit ecc errors from either cpu core
DBL.SELF	b01x1	count double-bit ecc errors from 'this' cpu core

Table 4-121. Unit Masks for SI_WDQ_ECC_ERRORS (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
ALL.EITHER	b1xx0	count all ecc errors from either cpu core
ALL.SELF	b1xx1	count all ecc errors from 'this' cpu core

SI_WRITEQ_INSERTS

- **Title:** Write Queue Insertions
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xa1, **Max. Inc/Cyc:** 2, **MT Capture Type:** S
- **Definition:** Counts insertions into the WRITEQ. Qualified by the following Unit Mask.

Table 4-122. Unit Masks for SI_WRITEQ_INSERTS

Extension	PMC.umask [19:16]	Description
ALL.EITHER	b0000	count all types of insertions into the write queue from either cpu core
ALL.SELF	b0001	count all types of insertions into the write queue from 'this' cpu core
IWB.EITHER	b0010	count implicit write back insertions into the write queue from either cpu core
IWB.SELF	b0011	count implicit write back insertions into the write queue from 'this' cpu core
EWB.EITHER	b0100	count explicit write back insertions into the write queue from either cpu core
EWB.SELF	b0101	count explicit write back insertions into the write queue from 'this' cpu core
WC1_8A.EITHER	b0110	count WC (size 1 thru 8 bytes) insertions into the write queue from either cpu core (addr[3] = 0)
WC1_8A.SELF	b0111	count WC (size 1 thru 8 bytes) insertions into the write queue from 'this' cpu core (addr[3] = 0)
WC16.EITHER	b1000	count WC (size 16 bytes) insertions into the write queue from either cpu core
WC16.SELF	b1001	count WC (size 16 bytes) insertions into the write queue from 'this' cpu core
WC32.EITHER	b1010	count WC (size 32 bytes) insertions into the write queue from either cpu core
WC32.SELF	b1011	count WC (size 32 bytes) insertions into the write queue from 'this' cpu core
NEWB.EITHER	b1100	count 'nuked' explicit write back insertions into the write queue from either cpu core
NEWB.SELF	b1101	count 'nuked' explicit write back insertions into the write queue from 'this' cpu core
WC1_8B.EITHER	b1110	count WC (size 1 thru 8 bytes) insertions into the write queue from either cpu core (addr[3] = 1)
WC1_8B.SELF	b1111	count WC (size 1 thru 8 bytes) insertions into the write queue from 'this' cpu core (addr[3] = 1)

SI_WRITEQ_LIVE_REQ_HI

- **Title:** Write Queue Requests (upper bit)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xa3, **Max. Inc/Cyc:** 1, **MT Capture Type:** C
- **Definition:** Counts the number of live WRITEQ requests. The Montecito processor can have a total of 8 per cycle. The upper bit is stored in this counter.

Table 4-123. Unit Masks for SI_WRITEQ_LIVE_REQ_HI

Extension	PMC.umask [19:16]	Description
EITHER	bxxx0	transactions initiated by either cpu core
SELF	bxxx1	transactions initiated by 'this' cpu core

SI_WRITEQ_LIVE_REQ_LO

- **Title:** Write Queue Requests (lower three bits)
- **Category:** Front-Side Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xa2, **Max. Inc/Cyc:** 7, **MT Capture Type:** C
- **Definition:** Counts the number of live WRITEQ requests. The Montecito processor can have a total of 8 per cycle. The lower three bits are stored in this counter.

Table 4-124. Unit Masks for SI_WRITEQ_LIVE_REQ_LO

Extension	PMC.umask [19:16]	Description
EITHER	bxxx0	transactions initiated by either cpu core
SELF	bxxx1	transactions initiated by 'this' cpu core

SPEC_LOADS_NATTED

- **Title:** Number of speculative inter loads that are NaTd
- **Category:** Sphdeculation event **IAR/DAR/OPC:** Y/Y/N
- **Event Code:** 0xd9, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts number of integer speculative loads that have been NaTed. This event has following umasks

Table 4-125. Unit Masks for SPEC_LOADS_NATTED (Sheet 1 of 2)

Extension	PMC.umask [19:16]	Description
ALL	b0000	Count all NaT'd loads
VHPT_MISS	b0001	Only loads NaT'd due to VHPT miss
DEF_TLB_MISS	b0010	Only loads NaT'd due to deferred TLB misses
DEF_TLB_FAULT	b0011	Only loads NaT'd due to deferred TLB faults

Table 4-125. Unit Masks for SPEC_LOADS_NATTED (Sheet 2 of 2)

Extension	PMC.umask [19:16]	Description
NAT_CNSM	b0100	Only loads NaT'd due to NaT consumption
DEF_PSR_ED	b0101	Only loads NaT'd due to effect of PSR.ed

STORES_RETIRED

- **Title:** Retired Stores
- **Category:** Instruction Execution/L1D Cache Set 4 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xd1, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of retired stores, excluding those that were predicated off. The count includes integer, floating-point, semaphore, RSE, VHPT, uncacheable stores.
- **NOTE:** This is a restricted set 4 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5. Only ports 2 and 3 are counted.

SYLL_NOT_DISPERSED

- **Title:** Syllables Not Dispersed
- **Category:** Instruction Dispersal Events **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x4e, **Max. Inc/Cyc:** 5, **MT Capture Type:** A
- **Definition:** Counts the number of syllables not dispersed due to all reasons except stalls. A unit mask can break this down to 1 of 4 possible components.

Table 4-126. Unit Masks for SYLL_NOT_DISPERSED

Extension	PMC.umask [19:16]	Description
EXPL	bxxx1	Count syllables not dispersed due to explicit stop bits. These consist of programmer specified architected S-bit and templates 1 and 5. Dispersal takes a 6-syllable (3-syllable) hit for every template 1/5 in bundle 0(1). Dispersal takes a 3-syllable (0 syllable) hit for every S-bit in bundle 0(1)
IMPL	bxx1x	Count syllables not dispersed due to implicit stop bits. These consist of all of the non-architected stop bits (asymmetry, oversubscription, implicit). Dispersal takes a 6-syllable (3-syllable) hit for every implicit stop bits in bundle 0(1).
FE	bx1xx	Count syllables not dispersed due to front-end not providing valid bundles or providing valid illegal templates. Dispersal takes a 3-syllable hit for every invalid bundle or valid illegal template from front-end. Bundle 1 with front-end fault, is counted here (3-syllable hit).
MLX	b1xxx	Count syllables not dispersed due to MLX bundle and resteers to non-0 syllable. Dispersal takes a 1 syllable hit for each MLX bundle. Dispersal could take 0-2 syllable hit de pending on which syllable we re-steer to. Bundle 1 with front-end fault which is split, is counted here (0-2 syllable hit).
ALL	b1111	Count all syllables not dispersed. NOTE: Any combination b0000-b1111 is valid.

SYLL_OVERCOUNT

- **Title:** Number of Overcounted Syllables.
- **Category:** Instruction Dispersal Events **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x4f, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of syllables which were overcounted in explicit and/or implicit stop bits portion of SYLL_NOT_DISPERSED.

Table 4-127. Unit Masks for SYLL_OVERCOUNT

Extension	PMC.umask [19:16]	Description
---	bxx00	(* nothing will be counted *)
EXPL	bxx01	Only syllables overcounted in the explicit bucket
IMPL	bxx10	Only syllables overcounted in the implicit bucket
ALL	bxx11	syllables overcounted in implicit & explicit bucket

THREAD_SWITCH_CYCLE

- **Title:** Thread switch overhead cycles.
- **Category:** Thread Switch Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x0e, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts various cycle overhead related to thread switches.

Table 4-128. Unit Masks for THREAD_SWITCH_CYCLE

Extension	PMC.umask [19:16]	Description
---	bx000	(* nothing will be counted *)
CRAB	bx001	Cycles TSs are stalled due to CRAB operation
L2D	bx010	Cycles TSs are stalled due to L2D return operation
ANYSTALL	bx011	Cycles TSs are stalled due to any reason
PCR	bx100	Cycles we run with PCR.sd set
---	bx101	(* nothing will be counted *)
ALL_GATED	bx110	Cycles TSs are gated due to any reason **NOTE: THREAD_SWITCH_GATED event is available to monitor the number of times TSs have been gated.**
TOTAL	bx111	Total time from TS opportunity is seized to TS happens.

THREAD_SWITCH_EVENTS

- **Title:** Thread switch events.
- **Category:** Thread Switch Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x0c, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of thread switches and their causes.

Table 4-129. Unit Masks for THREAD_SWITCH_EVENTS

Extension	PMC.umask [19:16]	Description
MISSED	bx000	TS opportunities missed
L3MISS	bx001	TSs due to L3 miss
TIMER	bx010	TSs due to time out
HINT	bx011	TSs due to hint instruction
LP	bx100	TSs due to low power operation
DBG	bx101	TSs due to debug operations
---	bx110	(* count is undefined *)
ALL	bx111	All taken TSs

THREAD_SWITCH_GATED

- **Title:** Thread switches gated
- **Category:** Thread Switch Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x0d, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts the number of thread switches gated and their causes.

Table 4-130. Unit Masks for THREAD_SWITCH_GATED

Extension	PMC.umask [19:16]	Description
---	bx000	(* nothing will be counted *)
LP	bx001	TSs gated due to LP
---	bx010	(* nothing will be counted *)
---	bx011	(* nothing will be counted *)
PIPE	bx100	Gated due to pipeline operations
FWDPRO	bx101	Gated due to forward progress reasons
---	bx110	(* nothing will be counted *)
ALL	bx111	TSs gated for any reason

THREAD_SWITCH_STALL

- **Title:** Thread Switch Stall
- **Category:** Thread Switch Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x0f, **Max. Inc/Cyc:** 1, **MT Capture Type:** A
- **Definition:** Counts how many times the processor is stalled more than specified number of cpu cycles in umask field before deciding to do a thread switch. This is done by counting cycles from the last instruction retired to the point the processor decided to pend a TS.

Table 4-131. Unit Masks for THREAD_SWITCH_STALL

Extension	PMC.umask [19:16]	Description
GTE_4	b0000	>= 4 cycles (Any latency)
GTE_8	b0001	>= 8 cycles
GTE_16	b0010	>= 16 cycles
GTE_32	b0011	>= 32 cycles
GTE_64	b0100	>= 64 cycles
GTE_128	b0101	>= 128 cycles
GTE_256	b0110	>= 256 cycles
GTE_512	b0111	>= 512 cycles
GTE_1024	b1000	>= 1024 cycles
GTE_2048	b1001	>= 2048 cycles
GTE_4096	b1010	>= 4096 cycles
---	b1010 -b1111	(* nothing will be counted *)

UC_LOADS_RETIRED

- **Title:** Retired Uncacheable Loads
- **Category:** Instruction Execution/L1D Cache Set 3 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xcf, **Max. Inc/Cyc:** 4, **MT Capture Type:** A
- **Definition:** Counts the number of retired uncacheable load instructions, excluding those that were predicated off. It includes integer, floating-point, semaphores, RSE, and VHPT loads, and check loads (`ld.c`) which missed in ALAT and L1D (the only time this looks like any other load).
- **NOTE:** This is a restricted set 3 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5.

UC_STORES_RETIRED

- **Title:** Retired Uncacheable Stores
- **Category:** Instruction Execution/L1D Cache Set 4 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xd0, **Max. Inc/Cyc:** 2, **MT Capture Type:** A
- **Definition:** Counts the number of retired uncacheable store instructions. It includes integer, floating-point, RSE, and uncacheable stores. (only on ports 2 and 3).
- **NOTE:** This is a restricted set 4 L1D Cache event. In order to measure this event, one of the events in this set must be measured by PMD5.

§