Intel® Itanium® 2 Processor Reference Manual

For Software Development and Optimization

April 2003

Order Number: 251110-002

Contents

Figures

Tables

About this Manual 1

1.1 Overview

The Intel® Itanium® 2 processor is the second implementation of the Intel® Itanium® architecture and is available in the following varieties:

- Itanium® 2 Processor with 1.5M L3 Cache
- Itanium® 2 Processor 1GHz with 3M L3 Cache
- Itanium® 2 Processor 1.30GHz with 3M L3 Cache
- Itanium® 2 Processor with 4M L3 Cache
- Itanium® 2 Processor with 6M L3 Cache
- Low Voltage Itanium® 2 Processor

This document describes how the Itanium 2 processor implements features of the Itanium architecture, as well as specific features of the Itanium 2 processor that are relevant to performance tuning, compilation, and assembler programming. Unless otherwise stated, all of the restrictions, rules, sizes, and capacities described in this document apply specifically to the Itanium 2 processor and may not apply to other implementations of the Itanium architecture.

General understanding of processor components and explicit familiarity with Itanium 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*.

1.2 Contents

Chapter 2, "Itanium® 2 Processor Enhancements" compares the Itanium processor and the Itanium 2 processor, highlighting some of the considerations that should be taken when optimizing for the Itanium 2 processor.

Chapter 3, "Functional Units and Issue Rules" describes the number and type of available functional units, instruction issue rules, and heuristics for efficient instruction scheduling based upon machine resources and issue rules.

Chapter 4, "Latencies and Bypasses" describes latencies and bypasses for execution of the different instruction types on the Itanium 2 processor.

Chapter 5, "Data Operations" describes considerations for data operations such as speculative or predicated loads or stores, floating-point loads, and prefetches. Data alignment considerations are also discussed.

Chapter 6, "Memory Subsystem" provides an overview of the memory subsystem hierarchy on the Itanium 2 processor.

Chapter 7, "Branch Instructions and Branch Prediction" describes how hints for branch prediction and instruction prefetch are implemented on the Itanium 2 processor.

Chapter 8, “Instruction Prefetching” describes how prefetching is implemented on the Itanium 2 processor.

Chapter 9, “Optimizing for the Itanium® 2 Processor” is a summary that draws conclusions from important points noted in earlier chapters.

Chapter 10, “Performance Monitoring” discusses performance monitoring registers and implementations specific to the Itanium 2 processor.

Chapter 11, “Performance Monitor Events” summarizes the Itanium 2 processor events and describes how to compute commonly used performance metrics.

Chapter 12, “Model-Specific and Optional Features” discusses Itanium 2 processor model-specific behavior, such as executing CPUID instructions.

1.3 Terminology

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

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 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.

1.4 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*

1.5 Revision History

Revision Number	Description	Date
-001	Public release of the document.	June 2002
-002	Refresh to incorporate new Itanium® 2 processor models.	April 2003

Itanium® 2 Processor Enhancements 2

This chapter outlines the major differences between the Itanium 2 processor and the Itanium processor. This is not an exhaustive list, so a reference to more details accompanies each topic.

2.1 Implemented Instructions

The Itanium 2 processor implements the 64-bit long branch instruction (`brl`) instruction directly in hardware. This instruction was not implemented in the Itanium processor. It allows programmers to direct a branch to an address that uses all 64 address bits. Details on the `brl` instruction can be found in Section 3.3.2 and Section 7.2.1 in Volume 2 of the *Intel® Itanium® Architecture Software Developer's Manual*. There are some branch prediction performance implications associated with the `brl` instruction which are noted in Chapter 7, "Branch Instructions and Branch Prediction."

2.2 Functional Units and Issue Rules

In general, the Itanium 2 processor has more functional units than the Itanium processor.

- In particular, the Itanium 2 processor has 6 arithmetic logic units (ALUs) to perform arithmetic operations, compares, most multimedia instructions, etc. The Itanium processor can only issue four of these types of instructions per cycle.
- The Itanium 2 processor has four memory ports allowing two integer loads and two integer stores per cycle. The Itanium processor has two memory ports.
- The Itanium 2 processor can issue one SIMD floating-point (FP) instruction per cycle. The Itanium processor can issue two SIMD FP instructions per cycle.
- Under certain conditions, the Itanium 2 processor can issue I-type instructions to memory functional units, thus increasing the number of template pair types which can be issued in one cycle. For the Itanium processor, I-type instructions will only be issued to integer functional units.
- The Itanium 2 processor scoreboards multi-cycle operations such as first-level instruction cache (L1D) misses, multimedia, and floating-point operations.

  This means that when an integer operation uses the result of a multimedia operation and the integer operation is not scheduled to cover the latency, the dependent instruction group will wait until the multimedia data is available.

  A predicated off operation, with a use of a scoreboarded operand, will stall the issue group for one cycle if the predicate was generated in the previous cycle. A predicated off instruction with predicates generated two or more cycles earlier will not incur pipeline stalls even when operands are scoreboarded.

2.3 Operation Latencies

On the Itanium 2 processor, most latencies are the same or shorter than on the Itanium processor with a few exceptions, i.e. memory latencies are shorter, floating-point latencies are shorter. A few more bypasses exist which remove some asymmetries. Table 2-1, "Itanium® 2/ Itanium Processors

Operation Latencies" shows latencies for both the Itanium 2 processor and the Itanium processor. The areas of difference are indicated by non-shaded boxes. The two different latency numbers are separated by a forward-slash or '/'. When reading from left to right, the first latency number corresponds to the Itanium 2 processor and the second number corresponds to the Itanium processor.

2.4 Data Operations

2.4.1 Data Speculation and the ALAT

The Itanium 2 processor advanced load address table (ALAT) is fully associative while the Itanium processor ALAT is two-way associative.

On the Itanium processor, a `ld.c` which misses the ALAT causes a 10-cycle pipeline flush. On the Itanium 2 processor, the penalty is 8 cycles.

On the Itanium processor, if a `chk.a`, `chk.s`, or `fchkf` fails, an operating system (OS) handler will be invoked through a trap handler to steer execution to the recovery code at the location specified in the target field of the `chk.a/chk.s/fchkf` instruction. On the Itanium 2 processor, hardware will usually perform the resteer without operating system intervention. This reduces the resteer cost from approximately 200 cycles to 18 cycles. If any of the following conditions are not met, the Itanium 2 processor will trap to the OS to service the `chk.a/chk.s/fchkf`:

`psr.ic` = 1
`psr.it` = 1
`psr.ss` = 0
`psr.tb` = 0

If a `chk.a` follows a store within the same cycle, the `chk.a` will always fail on the Itanium processor. On the Itanium 2 processor, a 12-bit address compare against ALAT entries will occur. See Section 5.1, "Data Speculation and the ALAT" for more details.

2.4.2 Data Alignment

The Itanium processor can support misaligned integer accesses within 16-byte blocks; however, the Itanium 2 processor supports misaligned integer accesses within 8-byte blocks. Section 5.5, "Data Alignment" has greater detail on misaligned access support for the Itanium 2 processor.

Table 2-1. Itanium® 2/ Itanium Processors Operation Latencies

		Consumer									
		Qual. Pred.	Branch Pred.	ALU	Load Store Addr	Multi-media	Store Data	Fmac	Fmisc	getf	setf
Producer	Adder: add, cmp, tbit, addp4, shladd, shladdp4, sum, logical ops, 64-bit immed. moves, movl, post-inc ops (includes post-inc stores, loads, lfetches)	n/a	n/a	1	1/(1-2)[1]	3	1	n/a	n/a	n/a	1
	Multimedia	n/a	n/a	3	3	2	3	n/a	n/a	n/a	3
	getf	n/a	n/a	5/9	6/9	6/9	5/9	n/a	n/a	n/a	6/9
	setf	n/a	n/a	n/a	n/a	n/a	6/2	6/2	6/2	6/2	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	4/5	4/5	4/5	4/5	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	4/5	4/5	4/5	4/5	n/a
	INT 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	1/1	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
	Int Load[2]	n/a	n/a	N	N+1	N+1	N	N	N	N	N
	FP Load[3]	n/a	n/a	M+1	M+2	M+2	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,ar[4]	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[5]	n/a	n/a	D	D	D	D	n/a	n/a	n/a	D

1. On the Itanium® processor, the address computation instruction must be in an M-slot type to avoid an extra cycle of latency.
2. N depends upon which level of cache is hit. For the Itanium processor, N=2 for L1D, N=6 for L2, N=21 for L3. For the Itanium 2 processor, N=1 for L1D, N=5 for L2, N=(12-15) for L3. These are minimum latencies.
3. M depends upon which level of cache is hit. For the Itanium processor, M=8 for L2 and M=24 for L3. For the Itanium 2 processor, M=5 for L2 and M=(12-15) for L3. These are minimum latencies. The “+1” entries indicate one cycle is needed for format conversion.
4. Best-case values of C range from 2 to 35 cycles depending upon registers accessed. EC and LC accesses are 2 cycles. FPSR and CR accesses are 10-12 cycles.
5. Best-case values of D range from 6 to 35 cycles depending upon indirect registers accessed; Iregs pkr and rr accesses are faster at 6 cycles.

2.4.3 Control Speculation

The Itanium 2 processor implements features intended to increase the performance of applications by decreasing the cost for incorrect control speculation. There are two parts of the solution for the Itanium 2 processor:

- The first part allows speculative load operations (This includes `lfetch` without the `.fault` completer.) to abort and set a NaT bit at the time of a data translation lookaside buffer (TLB) miss. In contrast, the Itanium processor would wait for the hardware page walker (HPW) operation to complete the walk before setting the NaT bit.
- The second part allows for a `chk.s` instruction (also for a `fchkf/chk.a` instruction) to branch directly to the fix-up code without involving the OS. The Itanium processor faults on a `chk.s`, `chk.a`, or `fchkf` instruction and requests that the OS branch to the fix-up code.

Thus, deferrals on the Itanium 2 processor occur quickly and the branch to fix-up code occurs quickly.

The deferral at data TLB miss is turned off inside interrupt handlers (when `PSR.is` = 1), which allows `ld.s` and `lfetch` instructions to complete a TLB walk and possibly return data. Clearing the `dcr.dm` bit will also prevent speculative operations from deferring at data TLB miss. Fast deferral requires the `dcr.dm` bit to be set. Refer to Section 5.2, “Speculative and Predicated Loads/Stores” for more information.

2.5 Memory Hierarchy

Both the Itanium microarchitecture and the Itanium 2 microarchitecture incorporate a three-level cache structure. In general, line sizes of the Itanium 2 processor are twice as large as those of the Itanium processor. Also, latencies of the Itanium 2 processor are shorter that those of the Itanium processor. The third-level cache (L3) of the Itanium 2 processor is on-chip and runs at a higher core frequency, which results in a much shorter latency. The Itanium 2 processor has a two-level TLB design for both instruction and data, while the Itanium processor has a single-level instruction TLB. The Itanium 2 processor’s TLBs are larger. The following tables list some of the differences in caches and TLBs. Details can be found in Chapter 6, “Memory Subsystem.”

Table 2-2. L1I Cache Differences

	Size	Line Size	Associativity	Latency
Itanium® Processor	16 KB	32 bytes	4-way	1 cycle
Itanium® 2 Processor	16 KB	64 bytes	4-way	1 cycle
Low Voltage Itanium® 2 Processor	16 KB	64 bytes	4-way	1 cycle

Table 2-3. L1D Cache Differences

	Size	Line Size	Associativity	Latency	Write Policies
Itanium® Processor	16 KB	32 bytes	4-way	2 cycles	Write through, No write allocate

Table 2-3. L1D Cache Differences (Continued)

	Size	Line Size	Associativity	Latency	Write Policies
Itanium® 2 Processor	16 KB	64 bytes	4-way	1 cycle	Write through, No write allocate
Low Voltage Itanium® 2 Processor	16 KB	64 bytes	4-way	1 cycle	Write through, No write allocate

Table 2-4. L2 Unified Cache Differences

	Size	Line Size	Associativity	Integer Latency	Floating-point Latency	Write Policies
Itanium® Processor	96 KB	64 bytes	6-way	Minimum of 6 cycles	Minimum of 9 cycles	Write back, Write allocate
Itanium® 2 Processor	256 KB	128 bytes	8-way	Minimum of 5 cycles	Minimum of 6 cycles	Write back, Write allocate
Low Voltage Itanium® 2 Processor	256 KB	128 bytes	8-way	Minimum of 5 cycles	Minimum of 6 cycles	Write back, Write allocate

Table 2-5. L3 Cache Differences

	Size	Line Size	Associativity	Integer Latency	Floating-point Latency	Bandwidth
Itanium® Processor	4 MB or 2MB, off chip	64 bytes	4-way	Minimum of 21 cycles	Minimum of 24 cycles	16 bytes/cycle
Itanium® 2 Processor (1.5M)	1.5 MB, on chip	128 bytes	6-way	Minimum of 12 cycles	Minimum of 21 cycles	32 bytes/cycle
Itanium® 2 Processor (1 GHz, 3M)	3 MB, on chip	128 bytes	12-way	Minimum of 12 cycles	Minimum of 21 cycles	32 bytes/cycle
Itanium® 2 Processor (1.30 GHz, 3M)	3 MB, on chip	128 bytes	12-way	Minimum of 14 cycles	Minimum of 23 cycles	32 bytes/cycle
Itanium® 2 Processor (4M)	4 MB, on chip	128 bytes	16-way	Minimum of 14 cycles	Minimum of 23 cycles	32 bytes/cycle
Itanium® 2 Processor (6M)	6 MB, on chip	128 bytes	24-way	Minimum of 14 cycles	Minimum of 23 cycles	32 bytes/cycle
Low Voltage Itanium® 2 Processor	1.5 MB, on chip	128 bytes	6-way	Minimum of 14 cycles	Minimum of 23 cycles	32 bytes/cycle

Table 2-6. Instruction TLB Differences

	Hierarchy	Size	Associativity
Itanium® Processor	1 level: ITLB	64-entry	Full
Itanium® 2 Processor	2 levels: L1 ITLB, L2 ITLB	32-entry, 128-entry	Full, Full
Low Voltage Itanium® 2 Processor	2 levels: L1 ITLB, L2 ITLB	32-entry, 128-entry	Full, Full

Table 2-7. Data TLB Differences

	Hierarchy	Size	Associativity	Penalty for Missing First Level DTLB
Itanium® Processor	2 levels: L1 DTLB, L2 DTLB	32-entry, 96-entry	Direct, Full	10 cycles
Itanium® 2 Processor	2 levels: L1 DTLB, L2 DTLB	32-entry, 128-entry	Full, Full	2 cycles
Low Voltage Itanium® 2 Processor	2 levels: L1 DTLB, L2 DTLB	32-entry, 128-entry	Full, Full	2 cycles

2.6 Branch Prediction

The major differences in the Itanium 2 processor and the Itanium processor branch prediction support are:

- Latencies
- `brp` instructions are ignored for branch prediction, i.e. the `brp.imp` is not required to achieve zero-bubble branches.
- Indirect branch targets are predicted from the source branch register rather than from a hardware table.
- Possible reduced prediction of BBB bundles due to prediction encoding.
- More robust method for prediction structure repair after a mispredicted return.
- Hardware implementation of the `brl` (64-bit relative branch) instruction.
- Setting `ar.ec` = 1 is not required for perfect loop prediction.

Full details can be found in Section 7, “Branch Instructions and Branch Prediction.”

Table 2-8. Branch Prediction Latencies (in cycles)

	Itanium® 2 Processor	Itanium® Processor
Correctly Predicted Taken IP-relative Branch	0	1
Correctly Predicted Taken Indirect Branch	2	0
Correctly Predicted Taken Return Branch	1	1
Last Branch in Perfect Loop Prediction	0	2
Misprediction Latency	6+	9

2.7 Instruction Prefetching

The Itanium 2 processor has an improved implementation of streaming and hint prefetching. See Chapter 8, “Instruction Prefetching” for more details.

Functional Units and Issue Rules 3

This chapter describes the number and type of available functional units, instruction issue rules, and heuristics for efficient instruction scheduling based upon machine resources and issue rules.

3.1 Execution Model

The Itanium 2 processor issues and executes instructions in assembly order, so programmer understanding of stall conditions is essential for generating high performance assembly code.

In general, when an instruction does not issue at the same time as the instruction immediately before it, instruction execution is said to have *split issue*. When a split issue condition occurs, all instructions after the split point stall one or more clocks, even if there are sufficient resources for some of them to execute. Common causes of split issue in the Itanium 2 processor are:

- An explicit stop is encountered.
- There are insufficient machine resources of the type required to execute an instruction.
- Instructions have not been placed in accordance with issue rules on the Itanium 2 processor.

The Itanium 2 processor issues instructions in the order defined by the static schedule. Care should be taken by the code generator to avoid register dependencies within an issue group. The Itanium 2 processor does not insert implicit stop bits to break WAW hazards; thus, a WAW hazard between loads and stores will result in an 8-cycle penalty if the predicates are true. Other WAW hazards, such as those due to ALU operations, will result in non-deterministic results and also consider predicates.

Once instructions are issued as a group, they will proceed as a group through the pipeline. If one instruction in the issue group has a stall condition, the whole group will stall. This stall will also stall all instructions behind it (younger) in the pipeline.

3.2 Number and Types of Functional Units

Although parallel instruction groups may extend over an arbitrary number of bundles and contain an arbitrary number of each instruction type, the Itanium 2 processor has finite execution resources. If a parallel instruction group contains more instructions than there are available execution units, the first instruction for which an appropriate unit cannot be found will cause a split issue and break the parallel instruction group.

The front-end of the Itanium 2 processor pipeline can fetch up to two bundles per cycle and the back-end of the pipeline can issue as many as two bundles per cycle. Given that there are 3 instructions per bundle, the Itanium 2 processor can be considered a six instruction issue machine. For more on details on the pipeline, see Appendix A, “Itanium® 2 Processor Pipeline.”

The Itanium 2 processor has a large number of functional units of various types. This allows many combinations of instructions to be issued per cycle. Since only six instructions may issue per cycle, only a portion of the Itanium 2 processor’s functional units described below will be used each cycle.

There are six general-purpose ALU units (ALU0, 1, 2, 3, 4, 5), two integer units (I0, 1), and one shift unit (ISHIFT, used for general purpose shifts and other special instructions). A maximum of six of these types of instructions can be issued per cycle.

The Data Cache Unit (DCU) contains four memory ports. Two ports are generally used for load operations; two are generally used for store operations. A maximum of four of these types of instructions can be issued per cycle. The two store ports can support a special subset of the floating-point load instructions.

There are six multimedia functional units (PALU0, 1, 2, 3, 4, 5), two parallel shift units (PSMU0, 1), one parallel multiply unit (PMUL), and one population count unit (POPCNT). These handle multimedia, parallel multiply, and the `popcnt` instruction types. At most, one `pmul` or `popcnt` instruction may be issued per cycle. However, the Itanium 2 processor may issue up to six PALU instructions per cycle.

There are four floating-point functional units: two FMAC units to execute floating-point multiply-adds and two FMISC units to perform other floating-point operations, such as `fcmp`, `fmerge`, etc. A maximum of two floating-point operations can be executed per cycle.

There are three branch units enabling three branches to be executed per cycle.

All of the computational functional units are fully pipelined, so each functional unit can accept one new instruction per clock cycle in the absence of other types of stalls. System instructions and access to system registers may be an exception.

3.3 Instruction Slot to Functional Unit Mapping

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 issue ports: 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 define to which issue port the instruction is assigned. An instruction is mapped to a subset of the issue ports 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 issue port within that subset.

Table 3-1, "A-Type Instruction Port Mapping," Table 3-2, "I-Type Instruction Port Mapping," and Table 3-3, "M-Type Instruction Port Mapping" show the mappings of instruction types to ports and functional units. Section 3.3.2 describes the selection of the particular port based upon instruction position.

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 3-1. 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 3-2. 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	MM Popcount	popcnt	■	
I10	Shift Right Pair	shrp	■	
I11-I17	Extr, Dep Test Nat	extr{.u}, dep{.z} tnat	■	
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	■	■

Table 3-3. M-Type Instruction Port Mapping

Instruction Type	Description	Examples	Memory Port			
			M0	M1	M2	M3
M1, 2, 3	Integer Load	ld*sz*, ld8.fill	■	■		
M4, 5	Integer Store	st*sz*, st8.spill			■	■
M6, 7, 8	Floating-point Load	ldf*sz*, ldf*sz*.s, ldf.fill	■	■	■	■
	Floating-point Advanced Load	ldf*sz*.a, ldf*sz*.c.[clr | nc]	■	■		
M9, 10	Floating-point Store	stf*sz*, stf.spill			■	■

Table 3-3. M-Type Instruction Port Mapping (Continued)

Instruction Type	Description	Examples	Memory Port			
			M0	M1	M2	M3
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			■	

3.3.1 Execution Width

When dispersing instructions to functional units, the Itanium 2 processor views, at most, two bundles at a time with no special alignment requirements. This text refers to these bundles as the *first* and *second* bundles. A *bundle rotation* causes new bundles to be brought into the two-bundle window of instructions being considered for issue. Bundle rotations occur when all the instructions within a bundle are issued. Either one or two bundles can be rotated depending on how many instructions were issued.

3.3.2 Dispersal Rules

The Itanium 2 processor hardware makes no attempt to reorder instructions to avoid stalls. Thus, the code generator must be careful about the number, type, and order of instructions within a parallel instruction group to avoid unnecessary stalls. The use of predicates has no effect on dispersal – all instructions are dispersed in the same fashion whether predicated true, predicated false, or unpredicated. Similarly, `nop` instructions are dispersed to functional units as if they were normal instructions. The dispersal rules for execution units vary according to slot type; i.e. I, M, F, B, or L. The rules for the different slot types are described below.

Dispersal rules for F slot instructions:

- An F slot instruction in the first bundle maps to F0.
- An F slot instruction in the second bundle maps to F1.
- A SIMD FP instruction essentially maps to both F0 and F1. See Section 3.3.3 for more information on SIMD FP issue rules.

Dispersal rules for B slot instructions:

- Each B slot instruction in an MBB or BBB bundle maps to the corresponding B unit. That is, a B slot instruction in the first position of the template is mapped to B0; in the second position, it is mapped to B1; and in the third position, it is mapped to B2.
- The B instruction in an MIB/MFB/MMB bundle maps to B0 if it is a `brp` or `nop.b` and it is the first bundle, otherwise it maps to B2.
- For purposes of dispersal, `break.b` is treated like a branch.

Dispersal rules for L slot instructions:

- An MLX bundle uses ports equivalent to an MFI bundle. If the MLX bundle is the first bundle, the L slot instruction maps to F0. Otherwise, it maps to F1. However, there is no conflict when the MLX template is issued with an MMF or MIF bundle and the F op is a SIMD FP instruction.

Dispersal rules for I slot instructions:

- The instruction in the first I slot of the two-bundle issue group will issue to I0. The second I slot instruction will issue to I1.
- If the second I slot instruction can only map to an I0 port, see Table 3-2, an implicit stop will be inserted and the second I slot instruction will be issued in the next cycle. Thus, an I0-only instruction should be placed in the first I slot of a bundle pair. Only one I0-only instruction can be issued per cycle.
- An instruction in an I slot will not necessarily be issued to an I port. If the first two I slot instructions have been issued to the I ports, and an additional I slot instruction in the issue group contains A-type instructions as listed in Table 3-1, and M ports are available; these instructions will be mapped to available M ports. This allows the potential dual issue of the MII-MII bundle pair. This is new to the Itanium 2 processor and is not true on the Itanium processor.
- For the MLI template, the I slot instruction is always assigned to port I0 if it is in the first bundle or it is assigned to port I1 if it is in the second bundle. Thus, the bundle pair MII-MLI can never dual issue.

Dispersal rules for M slot instructions:

On the Itanium 2 processor, M slot instructions are grouped into four subtypes (see Table 3-3):

- Load subtype, which can be issued on either M0 or M1 or both (e.g. integer load, `sync`)
- Store subtype, which can be issued on either M2 or M3 or both (e.g. integer store, `alloc`, `getf`)
- Generic subtype, which can be issued on any of the four M ports (e.g. ALU, floating-point load)
- Special instructions, which can be issued on only M2 port (e.g. `getf`, `mov` to AR)

The issue logic can reorder M slot instructions between different subtypes but cannot reorder instructions within the same subtypes. For instance, within an issue group an integer store can precede an integer load without causing a split issue. The store will be mapped to M2 and the load to M0 since the two instructions were from different subtypes.

However, if a store precedes a `getf`, the store will be issued to M2 and a split issue will occur because the `getf` must issue on M2. Instructions within the same subtype cannot be reordered. Therefore, the code scheduler should place the `getf` instruction before the store to ensure the `getf` instruction is mapped to M2 and the store is mapped to M3 to avoid port oversubscription.

Dispersal becomes more complicated when generic subtype instructions early in the issue group consume M ports. There is no encompassing rule to cover these cases. **It is recommended that the more restrictive subtypes get scheduled first in the issue group**. Example 3-1 and Example 3-2 demonstrate some of the dispersal possibilities.

Note: M_A is a generic subtype, M_L is an integer load, and M_S is a store subtype instruction.

Example 3-1. M_AM_LI - M_SM_AI

The bundle pair M_AM_LI - M_SM_AI gets mapped to ports M2 M0 I0 - M3 M1 I1.

The first generic subtype instruction mapped to M2 causes the M_S instruction to be mapped to M3. If M_S is a `getf` instruction, a split issue will occur.

Example 3-2. M_AM_AI - M_SM_AI

The bundle pair M_AM_AI - M_SM_AI gets mapped to ports M0 M1 I0 - M2 M3 I1, which allows M_S to get the more favorable M2 port.

Table 3-4 shows the combination bundle types that the Itanium 2 processor can dual issue (indicated by the shaded areas). Rows contain first bundle pair; columns contain second.

Table 3-4. Dual Issue Bundle Types

	MII	MLI	MMI	MFI	MMF	MIB	MBB	BBB	MBB	MFB
MII										
MLI										
MMI										
MFI										
MMF										
MIB[1]										
MBB										

Table 3-4. Dual Issue Bundle Types (Continued)

	MII	MLI	MMI	MFI	MMF	MIB	MBB	BBB	MBB	MFB
BBB										
MBB										
MFB										

1. The B must be nop.b or brp

Note: Floating-point loads are generic subtype instructions. As such, the Itanium 2 processor can issue up to four per cycle. This capability is available to all normal and speculative floating-point loads of all sizes. Advanced floating-point loads, load pair instructions, and check load instructions are not generic and must issue on the two load ports while the floating-point stores only issue to the two store ports.

3.3.3 Split Issue and Bundle Types

Because there is an increased number of functional units in the Itanium 2 processor and I slot instructions can sometimes issue to M ports, many bundle pairs can dual issue. Resource oversubscription rarely occurs. Reasons that bundle pairs would not dual issue are explicit stops and dispersal problems mentioned in the previous section. In addition, there are several Itanium 2 processor-specific (rather than architectural) special cases that will cause split issue. These specific cases are listed below:

- Branches
  - — BBB/MBB — Always splits issue after either of these bundles.
  - — MIB/MFB/MMB — Splits issue after any of these bundles unless the B slot contains a `nop.b` or a `brp` instruction. A `br` instruction always introduces an implicit stop bit for these bundle types.
  - — MIB BBB — Splits issue after the first bundle in this pair from B port oversubscription.
- SIMD FP
  - — Only one FP instruction can issue per cycle if the instruction is an SIMD FP instruction. For instance, for the bundle pair MF_pI MFI, where F_p is a SIMD FP operation, there will be an implicit stop between the M and F instructions of the second bundle, even if the F instruction is a `nop.f`.
  - — Similarly, for the bundle pair MFI MF_pI, there will be an implicit stop between the M and F_p instructions of the second bundle since the F_p instruction must issue to the F0 port and the first F instruction has already mapped to F0.
  - — One case which might seem to cause a split issue, but does not, is the bundle pair MF_pI MLX. Even though the L slot acts like it maps to an F port, these two bundles can dual issue.

Latencies and Bypasses 4

This chapter describes latencies and bypasses for execution of the different instruction types on the Itanium 2 processor.

In general, integer instructions have one cycle of latency, floating-point instructions have four cycles of latency, multimedia instructions have two cycles of latency, and L1 cache hits have one cycle of latency. However, due to asymmetric bypasses, there are many special cases that need to be listed separately.

4.1 Control and Data Speculation Penalties

The Itanium 2 processor can compute the address of the recovery code from the offset in the `chk.a`/`chk.s`/`fchkf` instruction without having to trap to the OS fault handler. The speculative load recovery latencies listed in Table 4-1 are approximations based upon the time difference between the `chk.s`/`chk.a`/`fchkf` retirement and the completion of first instruction of the fix-up code. These latencies do not include possible cache or TLB latencies. Also, the cost of the recovery code itself is not included. Further information on advanced loads can be found in Section 5.1, "Data Speculation and the ALAT."

Table 4-1. Speculative Load Recovery Latencies

Instruction	Latency (cycles)
chk.a, both int and fp (ALAT hit), chk.s (no NaT/NatVal)	0
chk.a, both int and fp (ALAT miss), chk.s (NaT/NatVal)	18
ld*.c, ldf*.c (ALAT hit, L1/L2 hit)	0
ld*.c, ldf*.c (ALAT miss, L1/L2 hit)	8

4.2 Branch Related Latencies and Penalties

Table 4-2 describes latencies for branch operations and branch related flushes. See Section 7, "Branch Instructions and Branch Prediction" for more detailed information.

Table 4-2. Branch Prediction Latencies

Branch Type	Whether Prediction	Target Prediction	Front-end Bubbles
IP-relative	Correct	Correct	0
IP-relative	Correct	Incorrect	1/6[1]
Return	Correct	Correct	1
Return	Correct	Incorrect	6

1. The 6-cycle penalty is for IP-relative branches that cross a 40-bit boundary. Loop branches that are mispredicted take 7 cycles. These incur a full branch mispredict penalty.

Table 4-3. 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, tbit, addp4, shladd, shladdp4, 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	3	2	3	n/a	n/a	n/a	3
thash, ttag, tak, tpa, probe[1]			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+1	N	N	N	N	N
FP Load[4]	n/a	n/a	M+1	M+2	M+2	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

1. Since these operations are performed on the L1D, they interact with the L1D and L2 pipelines. These are the minimum latencies but they could be much larger because of this interaction.
2. Since these operations are performed on the L1D, they interact with the L1D and L2 pipelines. These are the minimum latencies which 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 L2, N=12-15 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 L2, M=12-15 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. Iregs pkr and rr are on the faster side being 6 cycle accesses.

4.3 Latencies for OS Related Instructions

Table 4-4 lists the latencies for accesses to the CR, AR, and KR registers and the serialization latencies associated with many driver or OS operations, such as virtual address creation.

Table 4-4. Latencies for OS Related Instructions

READ Form			WRITE Form				
Register Type	Instruction	Latency	Instruction	Use	Latency	srlz.d	srlz.i
GR			mov r1=r1	int op	1		
			mov r1=imm22	int op	1		
			mov r1=imm64	int op	1		
FP			mov f1=f2	fp op	4		
PSR	mov r1=psr	12	mov psr=r1			6	17
	mov r1=psr.um	12	mov psr.um=r1	int op	5		
IP	mov r1=ip	2	READ ONLY				
PR	mov r1=pr	2	mov pr=r1	int op	1		
			mov pr.rot=r1	int op	1		
BR	mov r1=br	2	mov br=r1	branch	7		
			mov br.ret=r1	return	7		
AR	mov r1=ar.kr0	12	mov ar.kr0=r1	read kr	1		
	mov r1=ar.rsc	12	mov ar.rsc=r1	loadrs	14		
	mov r1=ar.bsp	12			READ ONLY		
	mov r1=ar.bspstore	12	mov ar.bspstore=r1	flushrs	14		
	mov r1=ar.rnat	5	mov ar.rnat=r1	flushrs	3		
	mov r1=ar.ccv	11	mov ar.ccv=r1	cmpxchg	1		
	mov r1=ar.unat	5	mov ar.unat=r1	ld8.fill	6		
	mov r1=ar.fpsr	12	mov ar.fpsr=r1	fmac	7		
	mov r1=ar.itc	36	mov ar.itc=r1	read itc	1		
	mov r1=ar.pfs	2	mov ar.pfs=r1	alloc	1		
				return	0		
	mov r1=ar.lc	2					
	mov r1=ar.ec	2					
CR	mov from CR0 (DCR)	12	mov to CR0 (DCR)			6	17
	mov from CR1 (ITM)	36	mov to CR1 (ITM)			35	
	mov from CR2 (IVA)	2	mov to CR2 (IVA)			7	
	mov from CR8 (PTA)	5	mov to CR8 (PTA)			6	17
	mov from CR9 (GPTA)	5	mov to CR9 (GPTA)			0	11
	mov from CR16 (IPSR)	12	mov to CR16 (IPSR)			6	
	mov from CR17 (ISR)	2	mov to CR17 (ISR)			7	
	mov from CR19 (IIP)	2	mov to CR19 (IIP)			7	
	mov from CR20 (IFA)	5	mov to CR20 (IFA)			6	
	mov from CR21 (ITIR)	5	mov to CR21 (ITIR)			6	

Table 4-4. Latencies for OS Related Instructions (Continued)

READ Form			WRITE Form				
Register Type	Instruction	Latency	Instruction	Use	Latency	srlz.d	srlz.i
	mov from CR22 (IIPA)	2	mov to CR22 (IIPA)			7	
	mov from CR23 (IFS)	12	mov to CR23 (IFS)			11	
	mov from CR24 (IIM)	2	mov to CR24 (IIM)			11	
	mov from CR25 (IHA)	5	mov to CR25 (IHA)			6	
	mov from CR64 (LID)	36	mov to CR64 (LID)			35	
	mov from CR65 (IVR)	36	READ ONLY				
	mov from CR66 (TPR)	36	mov to CR66 (TPR)			35	
	mov from CR67 (EOI)	36	mov to CR67 (EOI)			35	
	mov from CR68 (IRR0)	36	READ ONLY				
	mov from CR69 (IRR1)	36	READ ONLY				
	mov from CR70 (IRR2)	36	READ ONLY				
	mov from CR71 (IRR3)	36	READ ONLY				
	mov from CR72 (ITV)	36	mov to CR72 (ITV)			35	
	mov from CR73 (PMV)	36	mov to CR73 (PMV)			35	
	mov from CR74 (CMCV)	36	mov to CR74 (CMCV)			35	
	mov from CR80 (LRR0)	36	mov to CR80 (LRR0)			35	
	mov from CR81 (LRR1)	36	mov to CR81 (LRR1)			35	
IR	mov from cpuid[r0]	36	mov to cpuid[r0]			n/a	n/a
	mov from dbr[r0]	36	mov to dbr[r0]			1	
	mov from ibr[r0]	36	mov to ibr[r0]				46
	mov from msr[r0]	36	mov to msr[r0]			35	46
	mov from pkr[r0]	5	mov to pkr[r0]			11	22
	mov from pmc[r0]	36	mov to pmc[r0]			35	46
	mov from pmd[r0]	36	mov to pmd[r0]			35	46
	mov from rr[r0]	5	mov to rr[r0]			11	22

Data Operations 5

This chapter describes considerations for data operations such as speculative or predicated loads or stores, floating-point loads, and prefetches. Load hints, data alignment, and write coalescing considerations are also discussed.

5.1 Data Speculation and the ALAT

The family of instructions composed of `ld.a/ldf.a/ldfp.a`, `ld.c/ldf.c/ldfp.c`, and `chk.a` provide the capability to dynamically disambiguate memory addresses between loads and stores. Architecturally, the `ld.c` and `chk.a` instructions have a 0-cycle latency to consuming instructions. This allows the `ld.c/ldf.c/ldfp.c/chk.a` and the corresponding consuming instruction to be scheduled in the same cycle. However, if a `ld.c/ldf.c/ldfp.c/chk.a` misses in the ALAT, additional latency is incurred. Also, an advance load activates the scoreboard for the target register in order to ensure correct operation in the event of a L1D miss.

A `ld.c,ldf.c,` or `ldfp.c` that misses the ALAT initiates an L1 cache access. Other instructions in the issue group will be re-executed. This is an 8-cycle penalty that will affect all operations issued since the check load, whether there was a consumer in the same issue group or not. The consumer will be exposed to any additional cache latency (i.e. if the check load is found in the L1 then the penalty will be only 8 cycles). However, if the check load is in the L2, the user will see greater latency.

A `chk.a` that misses in the ALAT executes a branch to recovery code. On the Itanium 2 processor, the branch target can be computed from the offset contained in the `chk.a` instruction in most instances. This avoids the trap to the operating system that is done on the Itanium processor. The cost of a `chk.a` that misses in the ALAT is at least 18 cycles to branch to recovery code, plus the cost of the recovery code, plus the return. The actual resteer to fix up code occurs within 10 cycles, however there are at least 8 cycles for the first instruction of the fix up code to complete. The 8 cycles will increase when the branch to fix up code misses the L2 ITLB or L1I and other cache levels.

The Itanium 2 processor ALAT has 32 entries and is fully associative. Each entry contains the register number, type, and the lower 20 bits of the physical address. The address is used to compare against potentially conflicting stores while the register index and type support the check operation. Since only partial addresses are saved in the ALAT, it is possible to have a false conflict if a store and an ALAT entry had different addresses yet shared the same lower 20 bits. In addition, if a `ld.c` or `chk.a` follows a store too closely, the ALAT address comparison will be done on fewer than 20 bits. This is a result of the minimum 4K page size support and the need for both store and check addresses to be fully translated to accomplish the 20-bit comparison. Table 5-1 lists the distances and comparison sizes.

Note: Table 5-1, `ld.c` also implies `ldf.c` and `ldfp.c`.

Table 5-1. ALAT Entry Comparison Sizes

Distance	Comparison Size
st and ld.c in same cycle	12-bit
st precedes ld.c by 1cycle	12-bit
st precedes ld.c by more than 1 cycles	20-bit

Table 5-1. ALAT Entry Comparison Sizes (Continued)

Distance	Comparison Size
st and chk.a in same cycle	12-bit
st precedes chk.a by 1 cycle	12-bit
st precedes chk.a by more than 1 cycles	20-bit

Note: On the Itanium processor, if a store and `chk.a` occur in the same cycle, the `chk.a` will always fail, but this is not the case for the Itanium 2 processor.

5.1.1 Allocation/Replacement Policy

When a new entry is added in the ALAT, the following is the priority listing of which entry is replaced:

- The entry with the same register number as the new entry.
- The first invalid entry.
- A valid entry is replaced based upon advancing pointers associated with ports M0 and M1. This approximates a first-in - first-out (FIFO) algorithm.

5.1.2 Rules and Special Cases

The following rules and special cases should be noted:

- The Itanium architecture definition prohibits scheduling a `ld.a` and `ld.c` in the same cycle if both instructions have the same target register. Similarly, `ld.a` and `chk.a` cannot be scheduled in the same cycle if they have the same target register. However, separation by one or more cycles will give normal ALAT behavior. A similar situation is true for `ldf.a` and `ldfp.a`.
- A faulting `ld.a` will not write to the ALAT. Such faults are listed in *Volume 3: Instruction Set Reference* of the *Intel® Itanium® Architecture Software Developer's Manual* and include, among others, Data Page Not Present, Data TLB, and Unaligned Data Reference faults. In these situations, a subsequent corresponding `ld.c` or `chk.a` will definitely miss in the ALAT.
- If both an ALAT set and ALAT invalidate instruction occur in the same cycle, the ALAT set will not occur. For instance, if a `chk.a.clr rx` and `rx = ld.a[addr]` occur in the same cycle, the address of the `ld.a[addr]` will not be entered in the ALAT.

5.2 Speculative and Predicated Loads/Stores

Memory operations with speculative inputs behave in the following manner:

- For a normal load/store whose source register contains a NaT value, a register NaT consumption fault will occur.
- For a speculative load whose source register contains a NaT value, the NaT bit is set and a zero value will be returned.

The Itanium 2 processor supports two deferral behaviors: early and late. The behavior of speculative memory operations depends on several factors such as interrupt state, deferral control registers, and processor configuration. Early deferral mode is enabled through PAL procedure PAL_PROC_SET_FEATURES. The effects of this will be maintained until the system is rebooted and the processor returns to the default late deferral behavior. Table 5-2 lists the requirements to enable early deferral.

Table 5-2. Early and Late Deferral

Early Deferral Enabled	psr.ic	dcr.dm	Deferral Mode
Yes	0	0	Late
Yes	0	1	Early
Yes	1	0	Late
Yes	1	1	Late
No	x	x	Late

Table 5-3 shows the latency, according to deferral mode, that a speculative load may incur before returning data or eventually setting the destination NaT bit. The cost of each exception deferral ranges from one cycle to several cycles depending to the latency of the HPW. These HPW-related penalties cannot be scheduled around and affect every instruction in the issue group. Also, it is possible for the exception causing a deferral to not be resolved when the exception is deferred. Thus, the deferral stall may be seen each time through a loop where the `chk.s` is not reached.

Table 5-3. Control Speculation Penalties

Result	Hit/Miss	Penalty (early deferral)	Penalty (late deferral)
Return Valid Data	L1 DTLB Hit	1	1
	L2 DTLB Hit	4 + L2 latency	4 + L2 latency
	VHPT Hit	5 (no HPW walk)	20 + L2 latency
Set NaT Bit	NaT Source	1	1
	L1 DTLB Hit	2	2
	L2 DTLB Hit	2 or 4 + L2 latency	2 or 4 + L2 latency
	VHPT Hit	5 (no HPW walk)	22 or 2 + L2 latency
	VHPT Miss	5 (no HPW walk)	20 + L2 latency
	VHPT Fault	5 (no HPW walk)	17 + L2 latency

Note: Speculative loads are not limited to `ld.s` instructions. `lfetch` instructions are normally speculative and behave similarly to `ld.s` instructions with the exception that they never set a NaT bit or return data. An `lfetch` instruction may be made non-speculative with the `.fault` completer.

The advantage of early deferral is that speculative operations complete with low latency. The latency is at best three cycles for an early deferred `ld.s` as seen by a dependent operation. This is important in situations where the code generator is aggressive in its speculation and the chances of the speculative operation actually hitting in the data TLB is low. Since early deferral does not initiate a VHPT walk by the HPW, even valid requests may fault since they are not in the L2 DTLB.

5.3 Floating-Point Loads

Floating-point loads are not cached in the L1D and are instead processed directly by the L2. The limited size and bandwidth of the L1D makes caching this data unprofitable. It is expected that FP memory accesses can more easily be scheduled to cover the additional latency of the L2.

Floating-point loads incur an extra clock of latency over integer accesses to accommodate format conversion. Therefore, a floating-point load takes 6 cycles if it hits in L2. Note also, that the FP load pair instructions (both double-precision and single-precision) also access the L2 cache, so the latency for a load pair instruction is also 6 cycles assuming that it is an L2 hit.

5.4 Data Cache Prefetching and Load Hints

The architecture provides two software mechanisms to control when and where data is loaded. The `lfetch` instruction is used to explicitly prefetch data into the L1D, L2, or L3 caches. To facilitate more data locality, temporal hints can be used to control the level of the cache hierarchy into which loaded data is placed.

5.4.1 lfetch Implementation

The Itanium 2 processor implementation of `lfetch` is as follows:

- `lfetch.none` is completed only if there are no exceptions. Exceptions are not reported. Section 5.2 contains information on the behavior of `lfetch` instructions that encounter memory management faults.
- `lfetch.fault` is completed whether or not there is an exception. If there is an exception, it is raised to the OS to complete the operation. A TLB miss is resolved as with a normal load.
- If the `lfetch` misses in L1D but hits in the L2, the L1D cache is allocated based on the `lfetch` temporal hint. `lfetch` instructions have the same temporal locality behavior as integer loads.
- All `lfetch` types which miss in the first level data TLB and hit in the second level data TLB will stall the main pipeline and fill the first level data TLB as a normal load operation. The behavior of the `lfetch` in the event of an L2 DTLB miss depends on the use of the early or late deferral modes described in Section 5.2. In early deferral mode, the `lfetch` aborts with an L2 DTLB miss. In late deferral mode, the `lfetch` will initiate an HPW access. If the access fails, the `lfetch` will abort. However, it is only the `lfetch.fault` instruction that will initiate a HPW access when it misses both data TLBs.
- An `lfetch.excl` appears as a store to other cache levels and the system bus. This means that these operations will place a line in the M state within the caches. Do not use the `.excl` completer unless there is a high probability that the data will truly be modified. Otherwise, the cache will evict unmodified data to the cache structures and eventually to memory that is not modified.
- An `lfetch` to an uncacheable memory location will not reach the L2 cache as required by the architecture.

Note: The `lfetch` instruction appears as a load operation without a specific data return to the core. As such, many of the limitations that normal loads experience anywhere in the memory hierarchy will affect the `lfetch` instruction as well. Exceptions are noted and are provided with the intent that they will make `lfetch` instructions easier for the compiler to use in realizing performance.

5.4.2 Load Temporal Locality Completers

The Itanium architecture uses memory locality hints for managing the data cache hierarchy. On the Itanium 2 processor, four types of memory locality hints are implemented: `t1`, `nt1`, `nt2` and `nta`. The Itanium 2 processor does not support a non-temporal buffer; instead, non-temporal L2 accesses are allocated in L2 with biased replacement. The implementation is as follows:

- `t1` hint is for normal accesses. On a load, the line is allocated in L1D, L2, and L3. On a store, the line is allocated in L2 and L3, but not L1D.
- For loads with `nt1` hint, the line is only allocated in L2 and L3. In addition, the line is biased to be replaced in the L2. This is achieved by not updating the L2 LRU bits. Note that by doing so, the line has a higher probability of being replaced, though it is not guaranteed to be replaced next.
- Loads with `nt2` hint are implemented in the same manner as loads with `nt1` hint.
- For loads and stores with `nta` hint, the line is only allocated and biased to be replaced in L2. The line is not allocated into L3.

Table 5-4 lists how L1D, L2, and L3 handle line allocation and LRU update for different hints. Note that:

- L1D is write through and does not support FP loads and stores.
- The valid bit update in the L1D cache and the LRU bits update in the L3 cache are independent of the hint bits. Only the update of the L2 LRU is biased to mimic the behavior of a non-temporal buffer.

Table 5-4. Processor Cache Hints

Access	Hint	L1D		L2		L3	
		Alloc[1]	Update LRU Bits?	Alloc	Update LRU Bits?	Alloc	Update LRU Bits?
lfetch	t1	Yes	Yes	Yes	Yes	Yes	Yes
	nt1	No	No	Yes	Yes	Yes	Yes
	nt2	No	No	Yes	No	Yes	Yes
	nta	No	No	Yes	No	No	No
Integer load[2]	t1	Yes	Yes	Yes	Yes	Yes	Yes
	nt1	No	No	Yes	Yes	Yes	Yes
	nta	No	No	Yes	No	No	No
Integer store[3]	t1	No	No	Yes	Yes	Yes	Yes
	nta	No	No	Yes	No	No	No
FP load	t1	No	No	Yes	Yes	Yes	Yes
	nt1	No	No	Yes	No	Yes	Yes
	nta	No	No	Yes	No	No	No
FP store	t1	No	No	Yes	Yes	Yes	Yes
	nta	No	No	Yes	No	No	No

1. Alloc indicates an entry is allocated in that level of the cache on a cache miss.
2. Integer Load and FP Load - only t1, nt1, and nta attributes are allowed.
3. Integer Store and FP store - only t1 and nta are allowed.

Note: Other instruction/hint combinations are not allowed by the Itanium architecture.

5.4.2.1 General Descriptions of Hints

Memory locality hints are described below:

- `.none`: The load delivers the data and is loaded into both L1D and L2.
- `.nt1`: This hint means non-temporal locality in the first cache level capable of holding the referenced data. The Itanium architecture suggests this hint indicates that the load should deliver the data and the line should not be allocated in the first level caches. For the Itanium 2 processor, this hint will cause the line not to be allocated to the L1D on an integer cache miss. If it is already in the L1D cache, it will not be deallocated.
- `.nt2`: This hint means non-temporal locality in the second cache level capable of holding the instruction. For the Itanium 2 processor, this hint will cause integer accesses to the line to be allocated in L2; however, the LRU information will not be updated for the line (i.e. it will be the next line to be replaced in the particular set). If it is already in the L2 cache, it will not be deallocated.
- `.nta`: This hint means non-temporal locality in all levels of the cache hierarchy. For the Itanium 2 processor, this hint will cause the line to be allocated in L2; however, the LRU information will not be updated for the line (i.e. it will be the next line to be replaced in the particular set). This line will not be allocated in the L3 cache. If present in any cache, it will not be deallocated from that cache, although sometimes lines are deallocated for coherency reasons.

Note: There is no way to allocate only in L3 and not impact L2, even with an `lfetch` instruction.

The one-way allocation for non-temporal L2 data may lead to displacement of L2 data for a temporary data stream since the non-temporal data may be quickly replaced. A single L2 way holds 32KB. This may be large enough for a single `.nt` stream, but an attempt to use two non-temporal streams may cause one stream to displace the other.

5.5 Data Alignment

The Itanium 2 processor implementation supports arbitrarily aligned load and store accesses, except for integer accesses that cross 8-byte boundaries and any accesses that cross 16-byte boundaries.

If `psr.ac` = 1, all unaligned memory references will fault.

If `psr.ac` = 0, these rules must be followed to avoid faults:

- Integer loads and stores must be aligned within an 8-byte aligned window.
- All FP 4-byte and 8-byte load operations can be unaligned within a 16-byte aligned window.
- All FP load pairs must be naturally aligned; i.e. singles on an 8-byte alignment, doubles on a 16-byte alignment, `ldpr.8` on a 16-byte alignment.
- All FP 10-byte loads can be unaligned within a 16-byte window.
- FP fill/spill instructions must be aligned within a 16-byte aligned window.
- FP stores can be unaligned within a 16-byte aligned window.
- Semaphores (`cmpxchg`, `xchg`, `fetchadd`) must be restricted to natural alignment.
- All uncacheable (UC, WC) accesses which cross an 8-byte boundary will fault.

5.6 Write Coalescing

For increased performance of uncacheable references to frame buffers, previous generation IA-32 processors defined the write coalescing (WC) memory type. WC allows streams of data writes to be combined into a single, larger bus write transaction. The Itanium 2 processor fully supports write coalescing as defined by the Intel® Pentium® III processor. Like the Pentium III processor, the Itanium 2 processor WC loads are performed directly from memory and not from the coalescing buffers.

The Itanium 2 processor has a separate two-entry, 128-byte buffer (WCB) that is used for WC accesses exclusively. Each byte in the line has a valid bit. If all valid bits are true, then the line is said to be full and will be evicted by the processor.

5.6.1 WC Buffer Eviction Conditions

To ensure consistency with memory, the WCB is flushed on the following conditions (both entries are flushed). Table 5-5 shows the eviction conditions when the processor is operating in the Itanium system environment:

Table 5-5. Itanium® 2 Processor WCB Eviction Conditions

Eviction Condition	Itanium® Instructions
Memory fence (mf)	mf
Memory release ordering (op.rel)	st.rel, cmpxchg.rel, fetchadd.rel, ptc.g
Architectural Conditions for WCB Flush	
Flush cache (fc) hit on WCB	yes
Flush write buffers (fwb)	yes
Any UC load	no [1]
Any UC store	no [1]
UC load or ifetch hits WCB	no [1]
UC store hits WCB	no [1]
WC load/ifetch hits WCB	no
WC store hits WCB	no[2]

1. Itanium® architecture doesn't require the WC buffers to be coherent with respect to UC load/store operations.
2. A WC store which hits in the WCB updates that entry if it is not full. If it is full, a check is made if that entry is older or younger than the other WCB entry. If it is younger, the older WCB entry is flushed out (even if it is not full). The younger WCB entry is flushed afterwards. If the WCB entry is the oldest, it is flushed by itself.

5.6.2 WC Buffer Flushing Behavior

As mentioned previously, the Itanium 2 processor WCB contains two entries. The WC entries are flushed in the same order as they are allocated. That is, the entries are flushed in written order. This flushing order applies only to a "well-behaved" stream. A "well-behaved" stream writes one WC entry at a time and does not write the second WC entry until the first one is full. This implies that the addresses of the WC stores monotonically increase. A store with release semantics should be used to force a flush of a partial line before starting on the next line.

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

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

WCB eviction is performed for full lines by a single 128-bit bus transaction. For partially full lines, the WCB is evicted using 1-8, 16, or 32-byte transactions with the proper enables. The flushing will issue the largest data transactions allowed by a continuous and aligned set of write coalescing data. When flushing, WC transactions are given the highest priority of all external bus operations.

5.7 Register Stack Engine

The Itanium 2 processor register stack engine (RSE) only operates in lazy mode (`ar.rsc.mode` = 0). All other mode configurations are ignored.

A maximum of two loads or two stores can be performed by the RSE in each cycle, but not both loads and stores at the same time.

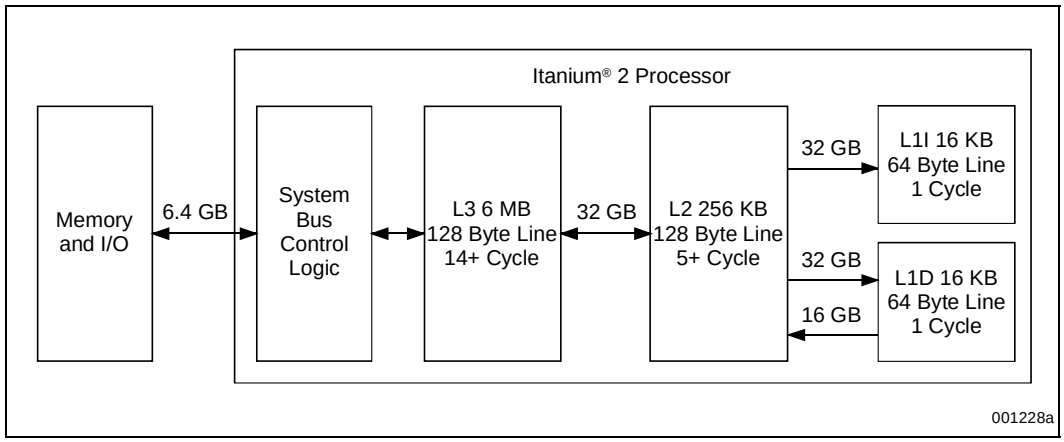
Generally, it is assumed that the RSE loads and stores will hit in the L1D cache and the L1D is capable of holding RSE cache lines in L1D.

Memory Subsystem 6

The Itanium 2 processor memory system has a three-level cache structure: a first-level instruction cache (L1I), a first-level data cache (L1D), a unified second-level cache (L2), and a unified third-level cache (L3).

The following sections contain detailed information on the workings of the L1D, L2, L3, and system bus. This information is presented to give a basis for the optimization recommendations. However, it is necessary to give enough understanding to recognize bottlenecks that are not specifically covered in this document. Chapter 9, "Optimizing for the Itanium® 2 Processor" provides some important suggestions in optimizing for the Itanium 2 processor memory subsystem.

Figure 6-1. Three Level Cache Hierarchy of the Itanium® 2 Processor



The Itanium 2 processor employs a two-level TLB for both instruction and data references: the first-level instruction TLB (L1 ITLB) and the second-level instruction TLB for instructions, and the first-level data TLB (L1 DTLB) and the second-level data TLB.

The Itanium 2 processor implements all the features of the Itanium architecture requirements for virtual memory support. Table 6-1 lists the specific parameters of the Itanium 2 processor implementation.

Table 6-1. Itanium® 2 Processor Virtual Memory Support

Virtual Memory	Itanium® 2 Processor Implementation
Page Size	4K, 8K, 16K, 64K, 256K, 1M, 4M, 16M, 64M, 256M, 1G, and 4G bytes
Physical Address	50 Bits
Virtual Address	64 Bits
Region Registers	8 registers with 24 bits in each register
Protection Key Registers	16 registers with 24 bits in each register

6.1 Translation Lookaside Buffers

Table 6-2 shows the major features of the TLBs of the Itanium 2 processor. The capabilities of the instruction and data TLBs are approximately equivalent. The first level TLBs are closely tied to the first level instruction and data caches. This is necessary to support the single cycle access for the L1 caches and comes at the price that a first level TLB miss forces a first level cache miss.

Table 6-2. Major Features of Instruction and Data TLBs

	Instruction TLBs	Data TLBs
Structures	L1 ITLB, L2 ITLB	L1 DTLB, L2 DTLB
Number of Entries	32, 128	32, 128
Associativity	Full, Full	Full, Full
Penalty for First Level Miss	2 cycles	4 cycles

6.1.1 Instruction TLBs

The L1 ITLB has 32 fully associative entries and is dual ported. One port is used exclusively for regular instruction fetches. The second port is shared among instruction prefetches, snoops, and TLB purges. The L1 ITLB contains sufficient information, region registers, and protection keys, such that it does not need to be a strict subset of the larger L2 ITLB.

When an L1 ITLB page translation is replaced, all entries in the L1I cache from the victimized page are invalidated. The victim entry is determined using true LRU. The L1 ITLB directly supports only a 4KB-page size. Other page sizes are indirectly supported by allocating additional L1 ITLB entries as each 4KByte segment of the larger page is referenced.

The L2 ITLB has 128 fully associative entries and is single ported. Up to 64 entries of the L2 ITLB can be assigned as translation registers (TRs). TRs are effectively translations locked into the L2 ITLB and are therefore not subject to LRU replacement policy. The L2 ITLB directly supports page sizes of 4KB, 8KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB, 64MB, 256MB, 1GB, and 4GB.

The L1 ITLB and L2 ITLB are accessed in parallel for demand fetches to reduce an L1 ITLB miss (and associated L1I cache miss) penalty. If an instruction access misses in the L1 ITLB, but hits in the L2 ITLB, the first-level instruction cache access will have two cycles of penalty (in parallel with the second-level cache latency) to transfer the page information from the L2 ITLB to the L1 ITLB. Since an L1 ITLB miss results in an L1I cache miss, the penalty will likely be greater as the instruction must be accessed from higher-level caches or the system memory.

6.1.2 Data TLBs

The L1 DTLB has 32 fully associative entries and is dual ported. Only two ports are required because it supports only integer load operations. Unlike the L1 ITLB, the L1 DTLB lacks protection and page attribute information. Consequently, the L1 DTLB is accessed in parallel with the DTLB and must be a strict subset of the second-level DTLB for an L1D hit.

When an L1 DTLB page translation is replaced, all entries in the L1D from the victimized page are invalidated. The L1 DTLB has a fixed page size of 4KB. Larger page sizes are supported by allocating additional L1 DTLB entries as a 4KB portion of the larger page.

The L2 DTLB has 128 fully associative entries and four ported. The four ports are needed to allow all combinations of integer loads, stores and floating-point loads to be looked up in parallel. The integer loads rely on the L2 DTLB for protection and page attribute information. The other accesses get virtual to physical mapping, protection, and page attributes from the L2 DTLB.

Up to 64 entries of the L2 DTLB can be assigned as TRs. TRs are effectively translations locked into the L2 DTLB and are therefore not subject to LRU replacement policy. The L2 DTLB directly supports page sizes of 4KB, 8KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB, 64MB, 256MB, 1GB, and 4GB.

Stores or floating-point accesses that miss the L1 DTLB incur no penalty from an L1 DTLB miss. Integer loads that miss the L1 DTLB but hit the L2 DTLB incur a 4-cycle penalty (in addition to the L2 cache latency) to transfer from the L2 DTLB to the L1 DTLB. Also, a load access that misses the L1 DTLB will not hit in the L1D.

6.2 Hardware Page Walker

The HPW is the third level of address translation. The HPW is an engine that performs page look-ups from the virtual hash page table (VHPT). When an L2 DTLB or L2 ITLB miss is encountered, the HPW will access (as necessary) the L2 cache, the L3 cache, and finally memory to obtain the page entry. If the HPW cannot locate the page entry in the L2, the L3, or memory, an interruption is generated and a software handler is called to complete the translation (unless the requesting instruction defers the exception). The HPW will accept a new instruction TLB miss when processing a data TLB miss (and visa versa); however, the HPW will not process them at the same time. The requests are effectively serialized.

Cache accesses must wait for TLB resolution to complete:

- L1D accesses both L1 DTLB and L2 DTLB in parallel.
- L1I accesses only require an L1 ITLB lookup (an L2 ITLB lookup is required upon an L1 ITLB miss).
- L2/L3 data access only require an L2 DTLB lookup.
- L2/L3 instruction accesses only require an L2 ITLB lookup.

When an L2 DTLB or L2 ITLB miss occurs, an HPW lookup is performed. This HPW walk may be aborted at any time. For non-speculative memory requests, when the HPW aborts or cannot successfully map the virtual address, a fault is raised. For speculative memory requests, the actual request is aborted and the `ld.s` will set the NaT bit. The minimum penalty for going to the HPW is summarized in Table 6-3. A HPW lookup does not look in or cause a fill of the L1D cache.

Since an L2 DTLB or L2 ITLB miss also implies a miss in the L1D or L1I, the penalty shown in Table 6-3 has the best case L2 cache latency added to the HPW walk latency.

Table 6-3. Best Case HPW Penalties

Event	Penalty in Cycles
Hit in L2	25
Miss in L2, hit in L3	31
Miss in both L2 and L3	20 + Main memory Latency (System dependent)

Table 6-3. Best Case HPW Penalties (Continued)

Event	Penalty in Cycles
HPW Abort	OS trap/abort
HPW mapping failed	OS trap/abort

6.3 Cache Summary

Table 6-4 summarizes the key parameters of the on-chip caches of the Itanium 2 processor.

Table 6-4. Cache Summary

	L1I	L1D	L2	L3
Size	16 KB	16 KB	256 KB	3 MB or 1.5 MB
Associativity	4-way	4-way	8-way	12-way
Line size	64 Bytes	64 Bytes	128 Bytes	128 Bytes
Latency	1 cycle	1 cycle	Minimum 5 cycles integer load use. Minimum 6 cycles floating-point load use. 7 cycles with 6 cycle stall penalty in *ROT* stage for instruction load use.	Minimum 12 cycles load use.
Tag Read Bandwidth	2 / cycle	4 / cycle	4 / cycle	1 / cycle
Data Read Bandwidth	1 X 32B / cycle	2 X 8B / cycle	2 x 16B / cycle + 2x 8B[1]	1 x 32B / cycle
Data banks	n/a	8 bytes/bank (store only)	16 bytes/bank	n/a
Write Bandwidth	n/a	2 x 8B / cycle	4 x 16B / cycle	1 x 32B / cycle
Fill Bandwidth	64 bytes assembly 2 cycles write - 1 cycle	64 bytes assembly 2 cycles write - 1 cycle	128 bytes assembly 4 cycles write - 1 cycle	128 bytes in 4 cycles
Outstanding Misses	7 prefetches	8 unique lines	16 unique lines	22 (16 read shared with L2, 6 write)
Line Size	64 Bytes	64 Bytes	128 Bytes	128 Bytes

1. The L2 read bandwidth is 48 bytes/cycle because the L2 can complete 2 `ldfpd` and 2 integer loads at a time. Any combination of 4 floating-point and integer returns may also complete every cycle.

6.4 First-Level Instruction Cache

The first-level instruction cache (L1I) is a 16KB, four-way set associative, physically addressed cache with a 64-byte line size. Lower virtual address bits 11:0, which represent the minimum virtual page, are never translated and are used for cache indexing. The L1I can fill a 64-byte line once every two cycles. It blocks on-demand fetch misses but is non-blocking for prefetch misses allowing up to seven to be outstanding to the L2 cache.

The L1I can sustain a rate of 32-byte reads per cycle to support a fetch rate of two bundles per cycle. The front-end always fetches aligned 32-byte bundle pairs from the L1I. If a branch points to the middle rather than the beginning of a 32-byte bundle pair, only the second bundle will be fetched. Therefore, branch targets must be on aligned 32-byte boundaries to achieve maximum fetch bandwidth from the L1I.

The tag array is dual-ported: one port is dedicated to instruction demand fetches, the other is shared between cache snoops and instruction prefetches. Cache snoops have priority over prefetches. The data array is dual ported, one for reading, one for fills. Additionally, special effort has been made to allow L1I reads and fills to occur simultaneously, thus there are few events that can keep an L1I miss from eventually writing into the L1I.

6.5 Instruction Stream Buffer

The Itanium 2 processor instruction stream buffer (ISB) is located between the L1I and the L2 caches. It serves as a line fill buffer for the L1I and assists in instruction prefetching. The ISB contains eight 64-byte cache lines or 8 double bundle pairs of instructions and is fully associative.

L1I lines returned from the L2, whether demand misses or prefetches, are all stored in the ISB. If a returned cache line is a demand miss, it will be forwarded to the instruction pipeline and may be moved into the L1I. The cache line remains in the ISB until an idle period where can drain into the L1I. The ISB entry may be victimized or invalidated before this move occurs preventing the L1I fill from occurring. The L1I supports both reads and fills at the same time, hence their ISB entries empty quickly into the L1I and few ISB victimizations or invalidations will occur.

The ISB is accessed in parallel with the L1I. An ISB hit has the same latency as an L1I hit. If the target line hits both the ISB and the L1I, the matching line in the ISB is invalidated.

6.6 First-Level Data Cache

The first-level data cache (L1D) is a multi-ported, 16KB, four-way set associative, physically-addressed cache with a 64-byte line size. The L1D is non-blocking and in-order. Lower virtual address bits 11:0, which represent the minimum virtual page, are never translated and are used for cache indexing.

The L1D is designed such that there are two dedicated load ports and two dedicated store ports. These ports are fixed, but the issue logic can rearrange loads and stores within an issue group to ensure they issue to the appropriate memory port. The load ports are dual ported, meaning that any two load addresses can be read from the memory in parallel without conflict. Stores, however, access the L1D data array in 8 groups that are 8 bytes wide. Stores do have the potential for conflicts, but the store buffer coalescing hardware limits the impact such conflicts have on performance.

The access latency of the L1D is one cycle unless the use is for an address of another load operations (i.e. pointer chasing) in which case it is two cycles. The L1D enforces a write-through, with no write-allocate policy. All stores will go to the L2 cache whether they hit or miss in the L1D. If a store hits in the L1D, the data is kept in a store buffer until the data arrays become available to update the L1D. These store buffers are capable of merging store data and forwarding it to later loads with restrictions. The L1D allocates on load misses according to temporal hints, load type, and available resources.

The L1D is highly integrated into the integer data path. All integer loads must go through the L1D to return data to the integer register file/bypass network. Consequently, integer L1D misses, after being serviced by L2, L3, or memory, also use the L1D datapath to the integer register file and block any core load that may require the same L1D data path.

Floating-point loads do not access the L1D. This allows them to issue on any of the four memory ports with minimal restrictions. Floating-point load-pairs and any floating-point loads with ALAT interactions can only be dispersed on the load ports. Despite the fact that `lfetch` instructions do not deliver data to the core, they can only be issued on the two load ports because they may cause an L1D fill and that capability is only provided on the two load memory ports.

An unaligned data reference exception will be raised if an unaligned integer load crosses an 8-byte boundary. See Section 5.5, “Data Alignment” for more details about alignment support.

6.6.1 L1D Loads

When a core load request gets access to the L1D, it will access the L1D tag and data arrays at the same time. Rotators at the output of the L1D data array provide support for both little and big endian accesses as well as some unaligned accesses without penalty. A virtual to physical mapping must be in the L1 DTLB and L1D tags for a load request to be a L1D hit. If the load is a miss or is forced to miss the L1D, then the request is passed on to the L2 when there are sufficient resources. The miss may result in a L1D fill depending on resources and cache hints. At minimum, all L1D misses eventually update the target register. Floating-point loads and ordered operations are forced to miss the L1D, but will not cause an L1D fill.

The L1D has resources for up to 8 outstanding L1D fill-requests to the L2. If more than 8 misses are outstanding, the subsequent misses will be passed to the L2, but will not result in an L1D fill. If two or more accesses miss the L1D and are accessing the same L1D line, only one will request an L1D fill but will be passed to the L2 cache to be satisfied.

6.6.2 L1D Stores

All store requests are passed to the L2 cache since the L1D is a write through cache. A store that misses the L1D has no effect on the L1D. However, if the store is a hit, the L1D must update the data array so that later loads can see the new data. To support this, the store data is read from the source register and staged down the L1D pipeline. Each store pipeline (M2/M3) has independent store buffers and control logic.

When the data is ready to update the L1D data array, it is allowed to do so provided there are no conflicts. Other operations writing the data array at the same time, such as an L1D fill, a load accessing the same 8-byte bank, or a store to the same bank, may prevent the needed update. In this case, the store data is moved to a backup buffer and waits for the array to become available. The store buffer can coalesce younger stores accessing the same L1D 8-byte wide data bank. If the backup buffer cannot update the data array and is needed by a new store that it cannot coalesce, the L1D pipeline will stall to create an opportunity for the backup buffer to drain.

Given this organization, it may be better for stores targeting the same group to issue down the same L1D pipeline. For example, it would be better to have all accesses to bank 0 to issue down M2 and all accesses to bank 1 to issue down M3. Thus, when it comes time to update the array, M2 and M3 will not conflict and will be allowed to update without delay.

6.6.3 L1D Load and Store Considerations

Some memory requests may affect each other even when separated in time. This section covers some possible load/load, load/store, store/load, and store/store interactions for both the L1D hit and miss cases. Each discussion will have a summary and a suggested solution.

6.6.3.1 Load/Load Conflicts

Load requests that hit in the L1D have no conflicts with each other because the L1D is true dual ported. However, a load request that misses the L1D may have conflicts at the L2 due to bank conflicts. If low latency is needed, special care should be taken to avoid loads in the same issue group that access the same L2 bank, i.e. A[7:4] should be unique for L2 bound accesses.

A less obvious load/load conflict can occur when a load is waiting to issue to the L1D, but is preempted by an older load returning from the L2/L3 or system bus. Here, the older load is given priority and the younger load must wait. These events are difficult to predict and hence difficult to schedule around. However, the L2 cache will only take the M1 port if there is only one integer load to return in a cycle. Thus, a conflict can be avoided by not using the M1 port for loads. This should not be done if it adds to the critical path.

This same conflict may exist between loads and special requests that use the L2 data paths to get information to the core. These are the `probe`, `thash`, `ttag`, `tpa`, and `tak` instructions.

6.6.3.2 Load/Store Conflicts

A load and store conflict has very different implications depending on which occurs first, the load or the store. Despite the fact that issue groups are inherently parallel, loads and stores are ordered according to position in the issue group.

When a load precedes a store and the load is a hit, there are no conflicts. However, there are significant implications when the load precedes the store and they are both L1D misses. In this case, the load will miss the L1D and likely request an L1D fill. The store, if it is seen by the L1D before the fill associated with the load, will be an L1D miss. As such, the store will invalidate the L1D associated fill buffer entry and stop the L1D fill from occurring. This is necessary because there is no opportunity for the store to update the incoming data before the L1D fill. The Itanium 2 processor must ensure that a later load sees an earlier store, so the fill is cancelled and the merge of the store with the cache line is taken care of by the L2. If the fill occurs before the store, then the fill completes and a normal store update of the L1D is done. These statements are true if the load and store share A[49:6] (a full L1D cache line).

One method to avoid this issue is to place a use of the load result before a conflicting store. This ensures that the data is filled into the L1D. Once the L1D is filled, the store updates the L1D and proceeds on to the L2 cache. This suggestion may not be appropriate for single load accesses or when the L1D line is not accessed again after a conflicting store.

6.6.3.3 Store/Load Conflicts

When a store precedes a load, the store data must be seen by the load. In the case where the requests are L1D misses, the L2 ensures this occurs. When the operations are L1D hits, the response to the load depends on the common address bits and how many cycles separate the store and load.

Table 6-5 shows the different store/load penalties. The penalty may depend on whether the load accesses the same data as the store, a subset of the store's data, or is completely independent of the store.

Table 6-5. Store to Load Forwarding Penalties

Store Precedes Load	Loads Accesses Bytes Completely Within Store	Load Accesses Bytes Partially Within Store	Address Comparison
0 cycles	17 cycles	17 cycles	11:2
1 cycle	3 cycles	5 cycles	11:2
2 cycles	3 cycles	3 cycles	49:2
3 cycles	1 cycle	3 cycles	49:2

The 5 and 17 cycle penalties are due both to the load being forced to miss the L1D and to the load and store facing L2 conflict conditions. The 3 and 1 cycle penalties are due to the L1D recirculating the load request until the distance between the load and store exceeds 3 cycles. This allows time for the L1D to update the data array with the store data and allow the load to proceed as if there was no store.

To avoid store/load conflicts, the store and load must be separated by more than 3 cycles. If more than 3 cycles separation is difficult to achieve, then ensure at least 1 cycle separation.

6.6.3.4 Integer and Floating-Point Access Interactions

Floating-point loads and stores are passed directly to the L2 and bypass the L1D. If a floating-point store occurs to a line which is resident in the L1D cache, that L1D line will be invalidated. This can cause problems when integer and floating-point data share the same L1D cache line. This is possible when both integer and floating-point data exist in the stack or as part of the same data structure. Suppose that both an integer value and a floating-point value share the same 64-byte aligned block. An integer load will bring the line into the L1D. A later floating-point store will write to L2 and invalidate the L1D line. Thus, a subsequent load of the integer value will miss the L1D.

This may be mitigated by bringing the line back into the L1D through an `lfetch` after issuing the store or by using `.nt1` hints on the integer accesses to keep them from filling the L1D and scheduling them for L2 latency.

6.6.3.5 Store/Store Conflicts

The L1D is true dual ported for loads, but only pseudo-dual ported for stores; two stores cannot update the exact same location in the data array at the same time (see Section 6.6.2, “L1D Stores”). The store buffer design, with coalescing, prevents most store/store conflicts for L1D store hits. The exception is that two stores cannot update the same L1D bank at the same time. Should there be a conflict, the younger store will move into a store buffer and may later update the L1D data array without impacting the L1D pipeline. However, if the store buffer is unavailable, the L1D will stall until the store buffer is drained. The conflict does not exist if either of the two stores misses the L1D. Note that the two stores do not need to access the same L1D cache line to conflict.

6.6.4 L1D Misses

When an L1D request misses, it is passed on to the L2 once the L2 has sufficient resources available to hold the new request. The resources include at least an L2 OzQ entry and an L2 Data entry. A L2 Data entry must be available even for a load to be accepted. If either the L2 OzQ or Data is full, the operations and every other operation in the issue group will stall until these resources are made available.

The L2 control logic reserves some L2 OzQ entries to ensure that when a request is allowed to leave the L1D pipeline, there is an L2 OzQ entry available for it. The logic reserves four entries for every cycle of ambiguity which is three cycles. The result is that in some instances, such as streaming, only about 20 of the 32 L2 OzQ entries are available. The Itanium 2 processor only stalls the L1D pipeline when the L2 is full *and* there is a request in the L1D that needs to go to the L2.

6.6.4.1 L1D Forced Misses

There are some load instructions that are forced to miss the L1D. A floating-point load will always miss the L1D. An ordered load (`ld.acq`) is allowed to hit in the L1D, but if it does miss the L1D, all subsequent loads, regardless of address or ordering constraints, will be forced to miss the L1D until the L2 indicates that the ordered load is visible.

6.6.4.2 L1D Forced Invalidates

Just as some operations are forced to miss the L1D, some operations will invalidate the L1D. A floating-point store will invalidate the L1D if it is a L1D hit. Ordered stores and semaphores will also invalidate the L1D if they hit in the L1D to ensure that ordering is maintained.

6.7 Second-Level Unified Cache

The second-level unified cache (L2) cache is a unified, 256 KByte, 8-way set-associative cache with a line size of 128 bytes. The L2 tags are true four ported and are accessed as part of the L1D pipeline. The L2 employs write-back and write-allocate policies. The integer access latency to the L2 is 5, 7, 9 or greater cycles. Floating-point accesses take 6, 8, 10, or greater cycles, which includes the floating-point conversion stage. An L1I miss that hits in the L2 and uses the L2 5-cycle bypass incurs a 7-cycle latency with a 6-cycle stall penalty.

The L2 cache is non-blocking and out of order. All memory operations that access the L2 (L1D misses and all stores) check the L2 tags and allocate into a 32 entry queuing structure called the L2 OzQ. All stores require one of the 24 L2 data entries to hold data to eventually update the L2 data array. The operations issue, up to four at a time, to access the L2 data array when conflicts are resolved and resources are available. L1I instruction misses are also sent to the L2, but are stored in the Instruction Prefetch FIFO (IPF). The L2 OzQ and IPF requests arbitrate for access to the data array and the L3/system bus.

The L2 data array has 16 banks which are each 16 bytes wide. This allows for multiple simultaneous accesses provided each access is to a different bank. Floating-point loads may issue from the L2 OzQ and access the L2 data array four at a time since the L2 has four datapaths to the FP units and register file. The L2 does not have direct datapaths to the integer units and register file; integer loads deliver data via the L1D, which has two datapaths to the integer units and register file. Stores may issue from the L2 OzQ and access the L2 data array four at a time provided they are all to different banks.

The fill path width from the L2 to the L1D and the L1I is 32 bytes. The fill bandwidth from the L3 or memory to the L2 is 32 bytes per cycle. Four 32-byte quantities are accumulated in the L2 fill buffers, then the 128-byte cache line is written into the L2 in one cycle, thus updating both tag and data arrays. Note that an NRU algorithm is used for cache line replacement.

The L2 cache is not inclusive of the L1D and L1I caches. The L2 maintains state information for each line, tracking if the data stored is modified (M), exclusive (E), shared (S), invalid (I), or pending update (P). This allows the L2 to use the MESIP protocol to maintain cache coherency and track victimized lines.

6.7.1 L1D Requests to L2

Every cycle, the L1D may issue up to four requests to the L2. These requests may be L1D load/store misses, L2 recirculates, L2 fills, instruction fetches, or snoops. The L2 tags are true four ported and are part of the L1D pipeline. This allows all four L1D load or store requests to access the L2 tags and determine if they are an L2 hit or miss before being allocated into the L2 queuing structures. This feature allows L2 misses to be identified and quickly passed on to the system bus/L3. It also lowers the latency of L2 hit requests.

All L1D load, store, semaphore requests are placed in the L2 OzQ. All L1I instruction misses, which are issued through the L1D to the L2, are placed in the IPF where they arbitrate against the L2 OzQ for access to the L2 data arrays and the system bus/L3. Other requests coming from the L1D such as snoops and fills are transitory and are not queued.

Read (load) operations of the L2 data array occur two cycles before a write (store) of the L2 data array. This timing relationship becomes important when determining load/store data array conflicts.

The L2 provides 16 fill buffers to track L2 misses. Each L2 miss may result in modified data eviction. The L2 provides 16 victim buffers to hold victim data; however, only 6 L2 victims may be outstanding at a time.

6.7.2 L2 OzQ

The non-blocking nature of the L2 is made possible by the L2 OzQ. This structure holds up to 32 operations that cannot be satisfied by the L1D. These include all stores, semaphores, uncacheable accesses, L1D load misses, and L1D unresolved conflict cases. The L2 cache design requires fewer than 32 L2 OzQ entries to hold the maximum number of L1D requests in conflict-free cases. However, there are many conflict cases within the L2. These cases may increase request lifetimes in the L2 OzQ. Thus, the additional entries allow the L1D pipeline to continue to service hits and make additional requests of the L2 while the L2 resolves the conflicts. The conflicts increase the L2 latency and make L2 latency prediction impossible.

6.7.2.1 L2 OzQ Allocation and Deallocation

The L2 OzQ control logic allocates up to four contiguous entries per cycle starting from the last entry allocated the previous cycle. If there are too few entries available, the L1D pipeline is stalled to prohibit any additional operations being passed to the L2. Requests are removed from the L2 OzQ when they complete at the L2 - that is when a store updates the data array, when a load returns correct data to the core, or when an L2 miss request is accepted by the system bus/L3.

6.7.2.2 L2 OzQ Behavior

The L2 OzQ control logic enforces architectural ordering requirements; and in instances where the architecture allows, operations may complete out of order. An operation blocked due to conflict or issue restrictions does not block younger operations from completing. This allows for high resource utilization within the L2 resulting in a performance benefit. Additionally, the out-of-order

issue allows the L2 to quickly recover from circumstances where the L2 control logic was temporarily not able to retire requests.

The out-of-order and non-blocking nature of the L2 OzQ has the effect of removing any time relationships between operations. For example, if the code generator separates two operations by 4 cycles, they will appear 4 cycles apart in the L1D pipeline. However, conflicts may keep the first operation from issuing immediately and force it to wait in the L2 OzQ. This situation may result in the second operation actually completing in the L2 before the first operation, assuming no ordering restraints, despite their 4 cycles of separation in the code stream.

The latencies of the L2 hit accesses are typically 5, 7, or 9+ cycles. These several latencies arise from the fact that some operations can issue and access the L2 data array at different times depending on the resources required and what preceded the request. The lower latencies come from allowing L1D request to access the L2 data array before they are allocated in the L2 OzQ. These are the 5- and 7-cycle L2 OzQ bypass. All latencies listed as 9+ are for operations that cannot take these bypasses and must allocate into the L2 OzQ and then later issue from the L2 OzQ to access the L2 data array.

6.7.2.3 5- and 7-Cycle Bypass

New L1D requests may take the 5-cycle bypass of the L2 OzQ and issue directly to the L2 data array provided there are no conflicts with older operations in the L2 OzQ. This bypass may be granted to the entire issue group provided there are no conflicts within the issue group. If a conflict occurs, the older request will take the bypass while the younger requests may not. Semaphores will never take a 5 or 7 cycle bypass and have a minimum latency of 9 cycles.

L2 bank conflicts will be discussed in Section 6.7.3, but they are used here in an example of how the L2 re-orders request to give the lowest possible latency. Conflicts typically are due to multiple requests for the same L2 data array (bank conflict). Consider the an L1D request (issue) group below:

```
ldfs f20 = [0x004] (L2 Bank 0)
ldfs f21 = [0x008] (L2 Bank 0)
ldfs f22 = [0x00c] (L2 Bank 0)
ldfs f23 = [0x010] (L2 Bank 1)
```

The first load will take the 5-cycle bypass. The bank conflict between the first and second load will prohibit the second and third loads from taking the 5-cycle bypass. The fourth load will also take the 5-cycle bypass since there is no bank conflict with the older requests or architectural ordering requirements.

When a request is kept from taking the 5-cycle bypass, the next choice is the 7-cycle bypass. The bank conflict between the second and third load will keep the third load from taking the 7-cycle bypass.

The situation becomes more complicated when the instructions above are followed by more instructions to be satisfied by the L2. Consider the issue group of loads from the previous example which is immediately followed by the following issue group of loads:

```
ldfs f25 = [0x014] (L2 Bank 1)
ldfs f26 = [0x018] (L2 Bank 1)
ldfs f27 = [0x01c] (L2 Bank 1)
ldfs f28 = [0x020] (L2 Bank 2)
```

In this example, the f20 and f24 loads take the 5-cycle bypass. The f21, f22, and f23 loads will try to take the 7-cycle bypass. However, before they can take the bypass, the new request group with f25, f26, f27, and f28 comes along. In this issue group, f25 and f28 take the 5-cycle bypass. Doing

so blocks the older issue group from taking the 7-cycle bypass. Those requests must then issue from the L2 OzQ. This increases their minimum latency from 6 to 12 cycles. The latencies of the operations would be as follows (noted in parenthesis after the load):

```
ldfs f20 = [0x004] (6)
ldfs f21 = [0x008] (12)
ldfs f22 = [0x00c] (13)
ldfs f23 = [0x010] (6)
ldfs f25 = [0x014] (6)
ldfs f26 = [0x018] (13)
ldfs f27 = [0x01c] (14)
ldfs f28 = [0x020] (6)
```

6.7.2.4 L2 OzQ Issue

Every cycle the L2 OzQ searches for requests to issue to the L2 data array (L2 hits), the system bus/L3 (L2 misses), or back to the L1D for another L2 tag lookup (recirculate). See Section 6.7.3 for more information on L2 cancel conditions and Section 6.7.4 for more information on L2 recirculate conditions.

The L2 can issue up to four L2 hit accesses per cycle provided there are no conflicts among them or among earlier issued operations. The conflicts for L2 hits include L2 data array banks, register port, L1D fill, and ordering. In the case of the L1D fill, only one such load may issue. Also, since the L2 uses the L1D register return paths for loads, only two loads can issue per cycle.

The L2 can issue only one access to the system bus/L3 at a time. An L2 miss in the same L1D request group as an L2 hit should be on the M0 port to have the shortest L3 latency. If the miss is on another port, its latency will increase slightly.

The system bus/L3 control logic will then either accept or reject the request based on system bus/L3 resources and conflict cases. Once the request is accepted, it may be removed from the L2 OzQ. The L2 OzQ pipelines L2 miss requests; it does not wait for the system bus/L3 to accept a request before issuing another request.

6.7.3 L2 Cancels

The L2 cancels generally apply only to requests taking a 5 or 7 cycle L2 OzQ bypass. This is because in most cases, the issue logic considers the conflict cases and holds off issue until the conflict is resolved. The best example of holding off issue from the L2 OzQ are bank conflicts. All the information needed to avoid all possible issue time conflicts may not be available and some L2 OzQ issued requests must be later cancelled and re-issued. When an operation taking a bypass gets cancelled, it will re-issue from the L2 OzQ since the bypasses are only available to L1D request groups. When an L2 OzQ request is issued and then later cancelled, its latency will increase by four cycles.

The cancel logic may also cancel or block issue in more instances than expected due to issue logic simplification or unavailable information. For example, requests that are recirculated will be included in cancel/block calculations for other instructions considered for issue, or the issue logic will try to issue up to four requests that need to recirculate even though it cannot recirculate more than one request.

A 5 or 7 cycle bypass is more likely to be canceled for P3 operations because it is the youngest in the issue group and due to events external to the L2 such as System Bus/L3 returns and snoop requests. P0 requests are the least likely to be canceled because these are the oldest instructions in the issue group.

There are many reasons to cancel or block L2 OzQ issue. The reasons are placed into two categories: those that are predictably avoidable and those that are not.

6.7.3.1 Predictably Avoidable Cancel Conditions

L2 Data array conflicts: The data array has 16-byte wide banks. Bits 7:4 of the address determine the bank. Any requests with the same bank, regardless of cache line, are candidates for a bank conflict. Any L1D request group with multiple loads targeting the same bank will see the younger requests cancelled or the L2 OzQ issue blocked. This also applies to multiple stores targeting the same bank.

Since L2 loads and stores access the L2 data array at different times, a load and store in the same request group cannot have bank conflicts; however, there is potential for load and store bank conflicts between entirely different L1D request groups. Store requests access the data array three cycles after a load would. This means that a store issued at time X may block or cancel a load that would issue at time X+3 if they both access the same L2 bank.

The following examples show how the conflict logic considers the L2 data array access time to determine bank conflicts. The following two examples do not have bank conflicts:

```
ld8 r20 = [0x008] ;;
ld8 r21 = [0x010]
```

and:

```
st8 [0x008] = r20
ld8 r21 = [0x010]
```

However, the following example shows a bank conflict between the store and the last load, but not between any other requests:

```
st8 [0x000] = r0 ;;
ld8 r19 = [0x000] ;;
ld8 r20 = [0x008] ;;
ld8 r22 = [0x110]
```

Bank conflicts due to L1D fill requirements are slightly less predictable. These bank conflicts arise from the fact that an L1D fill requires 64 bytes of data and hence, four banks at a time. Additionally, the data path to the L1D can only support one fill every two cycles. These are not predictable because not all L1D misses will request an L1D fill. Section 6.6.1 has more information on which requests can require an L1D fill.

6.7.3.2 Unpredictably Avoidable Cancel Conditions

There are some bank conflicts that are generally unpredictable. These events are tightly coupled with the unpredictable events of system bus and L3 data returns. The unpredictable cancel conditions may result in unexplained L2 latency increases.

6.7.4 L2 Recirculate

The L2 OzQ will need to recirculate requests whenever the request does not have a clear indication of hit or miss, or the required resources to complete an L2 miss are unavailable.

The most predictable reason for a request to recirculate is that the request misses a line that is already being serviced by the system bus/L3, but has not yet returned to the L2. The L2 only retires L2 hits and primary L2 misses to an L2 line. It does not retire multiple L2 miss requests; additional misses remain in the L2 OzQ and recirculate until the tag lookup returns a hit. The request then

issues from the L2 OzQ and returns data (for a load) or updates the array (for a store) as a normal L2 hit request.

6.7.4.1 lfetch and Recirculation

There is one significant exception to this secondary L2 miss recirculate condition. `lfetch` instructions have been optimized to avoid allocation in the L2 OzQ if they meet the following criteria:

- Secondary access to an L2 miss.
- Will not fill the L1D.

Since these `lfetch` instructions are not allocated into the L2 OzQ, they cannot recirculate. The only way to guarantee that an `lfetch` instruction will not fill the L1D is to place temporal hints such as `.nt1`, `.nt2`, or `.nta`.

6.7.5 Memory Ordering

Itanium architecture memory ordering requires that a request with acquire semantics must reach visibility before any other younger operation. A request with release semantics must not reach visibility before older operations.

The L2 issue logic enforces the architectural release ordering semantic by blocking issue of a release request until it is the oldest operation in the L2 OzQ. The issue logic may issue a release operation that is not the oldest, but then cancel and re-issue.

If the ordered operation is not an L2 hit, the L2 control logic can speculatively make a system bus/L3 request of the line or transform the request to a prefetch. If the other L2 OzQ entries proceeding the ordered request do not conflict, the prefetch will have the benefit of starting the access early without violating ordering requirements. If there are conflicts, the request is re-issued to ensure proper ordering.

Since the L2 is responsible for maintaining architectural ordering, all loads that are in the shadow of a `ld.acq` must be seen by the L2. Thus, they are forced to miss the L1D until the `ld.acq` has achieved visibility.

6.7.6 L2 Instruction Prefetch FIFO

The Instruction Prefetch FIFO (IPF) is an 8 entry queue to hold L1I requests. Up to seven of these eight entries may contain prefetch requests. One slot is always reserved for a demand request. Just like the L2 OzQ, the IPF can have requests that are L2 hits, L2 misses, bank conflicts, or recirculates. The IPF faces the same issue restrictions for each of these requests as the L2 OzQ does. However, unlike the L2 OzQ hit requests, only one IPF L2 hit may be issued to the L2 data array per cycle. This is due to the fact that all IPF requests will return data to the L1I cache and the data path back to the L1I can only support one fill per cycle.

Since the L2 supports both instruction and data accesses, all L2 issue control logic chooses among instruction and data requests according to Table 6-6.

Table 6-6. L2 Issue Priorities

Priority	Request
1	Demand Instruction Fetch (IPF)
2	Demand Instruction Fetch (7 cycle bypass)
3	Data (L2 OzQ)
4	Data (7 cycle bypass)
5	Prefetch Instruction Fetch (IPF)
6	Prefetch Instruction Fetch (7 cycle bypass)
7	Data (7 cycle bypass)

6.7.7 L2 Load and Store Considerations

Some memory requests may affect each other even when separated in time. This sections covers some possible load/load, load/store, store/load, and store/store interactions for both the L2 cache. Since the L2 OzQ allows out of order issuing, the L2 OzQ will re-order requests to fully utilize the L2 data arrays in satisfying requests. As a result, any static timing place in the code stream may not have the desired result on L2 behavior, however there are still actions the code generator can take to increase performance.

6.7.7.1 Effective Releases

The L2 cache deals with load/store, store/load, and store/store conflicts by ensuring that the issue order in the L2 OzQ is the same as the program order of the operations. The L2 control logic leverages the architectural ordering mechanisms that already exist to address the possible conflicts.

When the L2 OzQ accepts a new request, it checks the physical address bit 49:2 against all older incomplete requests in the L2 OzQ. If a match exists and a conflict results, the control logic applies architectural release semantics to the incoming request. This is called *effective release*. The effective release association remains until the operation completes and causes the L2 issue and conflict logic to cancel the request until it is the oldest request in the L2 OzQ.

Table 6-7 summarizes the addresses and operation types that can experience an effective release.

Table 6-7. Effective Release Operations

Incoming Request	Matching Request	Effective Release
Load	Load	No
Load	Store	Yes
Store	Load	Yes
Store	Store	Yes

6.8 System Bus/L3 Interactions

All requests that the L2 cannot satisfy reach the system bus/L3 as a Read Line (RL) or Read For Ownership (RFO) request. The RL request is used for code and common load operations. The L2 may receive the line in M, E, or S for RL requests depending on L3 state or the snoop response provided on the system bus. The RFO request indicates the L2 intends to modify the line to store

data. Stores as well as `lfetch.excl` and `ld.bias` instructions result in Read For Ownership requests. These requests will always exist in the M state in L2. Table 6-8 summarizes this behavior.

Table 6-8. System Bus/L3 Requests and Final L2 State

L3/System Bus Request	L2 Request	L3 State			System Bus	
		S	E	M	HIT	No Hit
Read Line	Code Read	S	E	M	S	S
	Data Read	S	E	M	S	E
Read For Ownership	store	Miss	M	M	M	n/a
	lfetch.excl	Miss	M	M	M	
	ld.bias	Miss	M	M	M	

The L2 may make partial line requests of the system bus, but this is only for UC attribute accesses and is not part of this discussion because they are neither coherent nor a concern for performance.

The L2 will make one RL or RFO to the system bus/L3 per cycle. Each of these requests will have a dirty victim associated with it when the L2 way chosen for victimization is in the M state. The L2 issues a request to the system bus/L3 and then later confirms the request. This protocol exists to allow issuing requests to the system bus/L3 that are later cancelled and/or recirculated. The L2 may make a request, but will not confirm a request if there are insufficient resources available. The L2 will not issue two requests to the same L2 line. A request that is not confirmed will wait at least four cycles before it is issued again.

The system bus/L3 will decide if the request is accepted and inform the L2 based on address conflicts, available resources to support the read request and the associated dirty victim. The L2 will then deallocate the request from the L2 OzQ if the system bus/L3 accepts the request. An L2 request may be rejected (see Section 6.10). A rejected request will wait at least four cycles before it is issued again.

When the system bus/L3 is ready to deliver data to the L2, it will be indicated to the L2 and the L2 will prepare to receive the data. The data returns come 32 bytes (a chunk) at a time from the system bus/L3 with the critical chunk first. L3 returns have higher priority than system bus data returns and come consecutively. In many instances, an L2 miss may also cause an L1D fill. Since the L1D line width is only 64 bytes, there is sufficient data to cause an L1D fill when only two chunks have been received from the system bus or L3. These requests must access the L1D pipeline and may block core requests from entering the L1D pipeline during that cycle. If there are two L1D fills for an L2 miss, another fill will occur when the last two chunks have been received by the L2.

6.9 Third-Level Unified Cache

The third-level unified cache (L3) is a unified, 6 MByte, 24-way set associative cache with a 128-byte line size. Some versions of Itanium 2 processor may have L3 caches of 4, 3, or 1.5 MByte with set-associativity of 16, 12, and 6 way respectively. Latencies may also vary between the different cache sizes. See Chapter 2 for exact latency numbers. These caches are alike in all other respects.

All L3 accesses are for the entire 128 byte line – no partial line accesses are supported. The access latency is 12, 14, or more cycles. This latency depends on how quickly the L2 issues the request and the activity of the L3 at the time of the request.

On the Itanium 2 processor, L3 accesses are fully pipelined and thus have a much higher effective bandwidth than the L3 on the Itanium processor. The L3 tag array is single ported and is pipelined to allow a new tag access every cycle. The L3 data array is also single ported, but requires up to four cycles to transfer a full line of data to the L2 cache or to the system bus in the case of an L3 dirty victim.

The L3 is non-blocking and has an 8-entry queue to support multiple outstanding requests. This queue orders requests and prioritizes them among tag read/write and data read/write to achieve the highest performance given the operations required.

6.10 System Bus

The Itanium 2 processor system bus operates at 200 MHz and is comprised of multiple sub-busses for various functions, such as address/request, snoop, response, data, and defer. The data bus is 128 bits wide and operates source synchronously, achieving a peak bandwidth of 400 million memory transactions or 6.4 GB per second.

The system bus control logic is an In Order Queue (IOQ) and an Out of Order Queue (OOQ), which tracks all transactions pending completion on the system bus. The IOQ tracks the in-order phases of a request and is identical to all processors. The OOQ contents hold only a processors requests that are deferred. The IOQ can hold 8 entries while the OOQ can hold 18 requests which allows for a maximum of 19 transactions to be outstanding on the system bus from a single Itanium 2 processor.

L2 requests that have not been completed (i.e. have not accessed the L3 nor completed a data phase on the system bus) are maintained in structures of the following sizes:

- 16 outstanding read requests from L2.
- 6 outstanding dirty writeback requests from L2.
- 6 outstanding L3 writebacks (i.e. replacement of a *dirty* line) to be serviced by the main memory.
- A combination of 16 outstanding L3 writebacks or L3 *castouts* (i.e. replacement of a *clean* line depending on the coherence mechanism, this might incur memory traffic) to be serviced by the main memory.
- Two 128-byte coalescing buffers to support WC stores.

Read transactions (this includes store instructions that miss the L2) are placed in one of the 16 bus request queues (BRQs). Each of these may then be sent to the L3 to see if the L3 can satisfy the request. In the case where the request is also an L3 miss, the request is scheduled to generate a system bus request (either Bus Read Line or Bus Read Invalidate Line for stores). When the system bus responds with the data, the line is written to the L2 and L3 based on its temporal locality hints and type of access.

Branch Instructions and Branch Prediction 7

The Itanium 2 processor employs both static and dynamic methods for branch prediction. For static branch prediction, the Itanium 2 processor uses the hint completers from the branch instructions. For dynamic prediction, the Itanium 2 processor uses several hardware structures.

This chapter describes how branch prediction affects software execution. The front-end instruction fetching is decoupled from the back-end instruction execution through an 8-bundle instruction buffer. For more detail regarding the instruction buffer, see Appendix A, "Itanium® 2 Processor Pipeline." Throughout this chapter, the term 'bubble' refers to cycles for which the front-end cannot deliver useful data, because the penalty may never translate to a loss in performance if there is another event blocking the back-end from retiring instructions. In the case where the back-end is waiting for the front-end, the penalty is a stall.

Table 7-1, "Branch Prediction Latencies" summarizes branch prediction latencies for the Itanium 2 processor. Notice that in the case of a correctly predicted IP-relative branch, there is no front-end bubble.

Table 7-1. Branch Prediction Latencies

Branch Type	Whether Prediction	Target Prediction	Penalty
IP-relative	Correct	Correct	0 Front-end bubbles
IP-relative	Correct	Incorrect	1 Front-end bubble
Return	Correct	Correct	1 Front-end bubble
Return	Correct	Incorrect	6+ Pipeline stalls
Indirect	Correct	Correct	2 Front-end bubbles
Indirect	Correct	Incorrect	6+ Pipeline stalls[1]
Loop	Incorrect	N/A	7+ Pipeline stalls[1]
Any type	Incorrect	N/A	6+ Pipeline stalls[1]

1. The + refers to the fact that some branches may cause the front-end to stall. This is only for incorrectly predicted short (up to 16 bundles) forward branches. The additional latency will be at most 8 cycles and may be less depending on how many branches were seen by the front-end after the mispredicted branch was seen by the front-end.

The branch prediction microarchitecture in the Itanium 2 processor is significantly different from that of the Itanium processor. Branch prediction is closely tied to the L1I cache which allows for the zero bubble resteer.

Single-cycle branches experience a stall once every two cycles (i.e. a one-cycle loop takes four cycles to make three iterations). Single-cycle loops should be avoided. It is also possible that a stall may occur if several branches are encountered in succession. For example, if the front-end sees a branch every cycle for 3 cycles, one cycle of stall may occur.

7.1 Branch Prediction Hints

Information about branch behavior can be provided to the processor to improve branch prediction. This information can be encoded through branch hints as part of a branch instruction. Branch hints do not affect the functional behavior of the program and may be ignored by the processor.

Only hints specified within a branch instruction are used for branch prediction. Hints on the `brp` or `mov br` instructions are ignored by the branch predictor.

For the Itanium 2 processor, branch hints are `.sptk`, `.spnt`,`.dptk`, or `.dpnt` (`sp`=static prediction, `dp`=dynamic prediction, `tk`=taken, `nt`=not taken). The terms “static” and “dynamic” hints refer to the code generator’s confidence in the branch behavior. For example, `.sptk` means the code generator is very sure that the branch will be taken, whereas `.dptk` means that the code generator thinks the branch will be taken, but it is not so confident.

The impact of these branch hints depends on other branches in the two-bundle window and other branch information maintained in the processor. The consequence is that a branch with a `.dpnt` hint may be predicted taken the first time seen. The processor will quickly recover from this and correctly predict this branch in the future.

The use of `.dp`xx is recommended as default, unless the loop is a `ctop` or `cloop` in which case `.sp`xx is recommended.

The `.spxx` hint is also important for very short, 1 or 2 cycle, loops. With static prediction hints, these loops will not wait for the machine to generate a new hint prediction, but will instead use the take or not-taken from the static hint. If dynamic hints are used in the short loops, the processor may stall each iteration that the branch prediction requires updating.

The branch prediction hints have a an anomalous behavior when used in .bbb bundles. Normally, the branch hints of each branch instruction will effect only that specific branch. However, a `.bbb` bundle will always use the branch hints provided on the slot 0 branch for the slot 1 and slot 2 branches. There are a few ways to avoid this. The first is to break up the .bbb bundle into two other bundles. Unfortunately, this may not be good for code density and other solutions such as using a `.dpxx` hint or a `.spxx` with a .clr completer on the slot 0 branch should be considered.

7.2 Indirect Branches

The predicted targets of indirect branches, other than returns, are extracted from the source branch register of the indirect branch rather than from a hardware table. This has several implications.

There is always a penalty for indirect branches on the Itanium 2 processor. A two-cycle front-end bubble is seen for a correctly predicted indirect branch. An incorrect taken/not taken or address prediction is 6 or more pipeline stalls. The address prediction is based on the contents of the branch register referenced by the branch as seen by the front-end. An in-flight update to the branch register will not be seen by the front-end and the predicted target may be wrong. Correct target prediction requires that the branch register write precede the indirect branch by several cycles. This distance varies since the front and back-ends of the pipelines are decoupled. A code generator can minimize the impact of this in the following ways:

- Separate the write and indirect branch by at least 6 front-end L1I cache accesses.
- Add an additional write to the branch register above the true branch register writer to hint the target.

- Use different branch registers for each indirect branch instance to minimize conflicts with other indirect branches.

7.3 Perfect Loop Prediction

In many cases, the perfect loop predictor can correctly predict the back-edge branch of a counted loop, i.e., `cloop` or `ctop` type branches, including the fall-through instance, as well as the loop back iterations. Unlike the Itanium processor, the Itanium 2 processor does not need `brp` to accomplish this.

The Itanium 2 processor uses the PLP only for the final iteration of the loop. The initial loop predictions are decided on dynamic or static information based on the hints used.

If the last branch of a loop is predicted correctly, there might still be a one- or two-cycle bubble in order to get this correct prediction. The smaller the number of loop iterations, the more likely it is that there will be a two-bubble resteer. Conversely, the larger the loop iteration, the more likely it is that there will be a zero-bubble resteer. The PLP uses the current values of `ar.lc` and `ar.ec` for prediction, so any writers to these registers should be well ahead of the counted loop branch to assure correct prediction.

In some instances, the Itanium processor required that `ar.ec` be set to 1 for correct prediction. The Itanium 2 processor does not have this same requirement and actually expects `ar.ec` = 0 when there is no epilog.

Instruction Prefetching 8

The Itanium 2 processor supports several forms of instruction prefetching. Instruction prefetch is defined to be the act of moving instruction cache lines from higher levels of cache or memory into L1I. Streaming prefetching initiates hardware prefetching of the next cache lines, either sequential or at the target of predicted taken branches. Hint prefetching allows software to specify a particular line, or lines, to be prefetched. On the Itanium 2 processor, it is expected that instruction prefetching will be an effective way to reduce instruction cache misses since the code generator has a wide degree of control over the prefetch agent and the Itanium 2 processor cache design specifically considered prefetching.

8.1 Streaming Prefetching

Streaming prefetching is initiated by using the `.many` completer on branch instructions. If the front-end processes a branch with a `.many` completer, the prefetch engine will continuously issue prefetch requests, at one request per cycle, for subsequent instruction lines, into the prefetch pipeline. The prefetch request is checked against the L1I and the L1 ITLB. If it hits in the L1 ITLB and misses in the L1I, the request is sent to the L2, otherwise it is discarded. The lines are prefetched starting at the branch target plus 64 or 128 bytes (depending on the alignment of the branch target). Streaming prefetching continues until one of the following stop conditions occurs:

- A predicted-taken branch is encountered by the front-end
- A branch misprediction occurs
- A `brp` instruction without the `.imp` completer is encountered by the front-end[1]

The L1I cache design allows both fill and lookups to occur at the same time. Thus, the lifetime of a request in the ISB is typically very small. This allows the prefetch engine to prefetch instructions with little chance that the line will get overwritten before it is used. If the branch is predicted taken by the front-end, prefetching will be initiated in the front-end. If the branch is incorrectly predicted not-taken by the front-end, prefetching will be initiated by the back-end when the prediction is corrected. However, if the opposite case occurs, i.e. the branch is incorrectly predicted taken in the front-end, prefetching will be terminated and it will NOT be restarted when the back-end corrects the prediction. Finally, if the branch is incorrectly predicted-taken by the front-end, prefetching will be terminated when the prediction is corrected by the back-end.

A `.many` prefetch stream may be halted by an L1I TLB miss. The event does not cancel the prefetch, but suspends the prefetch until the L1I TLB fill completes at which point the prefetch continues until stopped from one of the reasons described above.

1. A `brp` instruction suggests that an associated br.many is *around the corner*. The assumption is that the prefetch engine has already prefetched past the br.many, and additional prefetches would be useless. The reason that a brp.imp does not terminate prefetching is related to Itanium® processor code. In the Itanium processor, brp.imp instructions are used to predict branches and might not have any association with a br.many.

Table 8-1. Summary of Streaming Prefetch Actions

	Predicted Taken	Predicted Not-Taken
Actually taken	Any current streaming prefetch is stopped in the front-end. If the branch has a *.many* completer, a new stream is started by the front-end.	Any current streaming prefetch is stopped in the back-end. If the branch has a *.many* completer, a new stream is started in the back-end.
Actually not-taken	Any current streaming prefetch is stopped in the front-end. It is NOT restarted when the misprediction is detected. If branch has a *.many* completer, a new stream is started in the front-end. It is terminated when the misprediction is detected by the back-end.	No effect on any current streaming prefetch. A new stream in NOT started.

8.2 Hint Prefetching

Hint prefetching is initiated with the `brp` or `mov br` instructions. Unlike the Itanium processor, the Itanium 2 processor prefetch initiation does not affect branch prediction state. However, it has this same restriction as the Itanium processor: `brp` instructions must be on the last instruction slot (slot 2) of a bundle in order to be processed; otherwise, it is ignored. `brp` instructions have no associated branch prediction effects. Table 8-2 illustrates the prefetching mechanisms associated with the branch hints.

Table 8-2. Prefetch Mechanisms

Branch Hint	Prefetch Mechanism
brp.(sptk,loop,dptk).few	Normal prefetch of 1 cache line generated.
brp.(sptk,loop,dptk).many	Prefetches 2 cache lines from target.
brp.(sptk,loop,dptk).imp.few	Flushes prefetch virtual address buffer (PVAB) and prefetches 1 cache line.
brp.(sptk,loop,dptk).imp.many	Flushes prefetch virtual address buffer (PVAB) and prefetches 2 cache lines.
move_to_br.(sptk,dptk).few	All other fields ignored, prefetches 1 cache line.
.many hint	Streaming prefetches triggered off predicted taken IP-relative branches.

A `.few` completer will prefetch one-half or one L2 line, depending on the alignment of the associated branch target, and a `.many` completer will prefetch 1.5 or 2 L2 lines, depending on the alignment of the associated branch target. Hint prefetches are sent to the 8-entry prefetch virtual address buffer (PVAB). Up to 2 hint prefetches can be sent to the PVAB in each cycle.

In a given cycle, if the prefetch pipeline is not stalled and if a `br.many` is not active, a prefetch request is removed from the PVAB. The prefetch request is then checked against the L1I and the L1 ITLB. If it hits in the L1 ITLB and misses in the L1I, it is sent to L2, otherwise it is discarded. The intent is to use hint prefetches to prefetch the first “chunk” of instructions at the target of a branch and to use streaming prefetching to prefetch the subsequent instructions. In order to fully hide the latency of an L2 hit, a hint prefetch should precede a branch by 9 fetch cycles. If a `br.many` is preceded by a `brp.many`, there will be some overlap between the prefetches generated by the two instructions. While this overlap is wasteful, there is benefit in having more lines prefetched earlier (as opposed to presaging the `br.many` by a `brp.few`). `brp.few` prefetches might be useful in conjunction with streaming prefetches as described in Section 8.1.

8.3 Prefetch Flush Hints

Certain forms of `brp` instruction have the side effect of flushing the contents of the PVAB and possibly the prefetch pipeline. These are provided to give the compiler some control over the state of prefetching.

- `brp.few.imp` - will remove all `brp.few` prefetches from the PVAB (but not any already in the prefetch pipeline).
- `brp.exit.imp` - will remove all prefetches from the PVAB and those in the prefetch pipeline, and additionally will stop the streaming prefetch engine (and therefore will stop `br.many`, `brp.many` and `brp.exit` prefetching).
- `brp.*.imp` - (`brp` without the `.imp` completer) will cancel any streaming prefetches initiated by a `br.many` instruction.The intent is to allow the compiler to stop a `br.many` from prefetching too far.

The flushing side effect is in addition to the normal behavior of these prefetch instructions. Note that flushing a prefetch once it reaches the pipeline may not be effective (i.e. the prefetch may still be issued to the L2 and beyond).

8.4 The brl Instruction

The Itanium 2 processor implements the `brl` instruction that provides 64-bit relative branches. These long relative branch instructions have less cost than in the Itanium processor, but they are higher cost than the short relative branch `br` instructions. Specifically, the branch prediction mechanisms in the Itanium 2 processor do not calculate the predicted target correctly for `brl` instructions unless it is set when the L1I cache line is written. Thus, if a `brl` prediction target is aliased with another branch in the bundle pair, the target will be incorrect and the branch will see a full branch mispredict penalty and it will not be fixed.

The `brl` instruction is much more efficient than multiple short jumps despite this cost. However, The linker should place `brl` instructions only where they are specifically needed.

Optimizing for the Itanium® 2 Processor 9

This chapter is a summary of conclusions that can be drawn from important points noted in earlier chapters. These guidelines are not applicable in all situations and profiling should be used to guide the use of optimizations.

9.1 Hints for Scheduling

Observing the following heuristics whenever possible will minimize the chances of implicit stops or unexpected dispersal related stalls:

- Schedule the most restricted instructions early in the bundle. This lessens the chance that a generic subtype instruction will consume a port which is needed by a later more restricted instruction.
- In some cases, placing A-type instructions in I slots rather than M slots might achieve denser bundling. If this is done, place any I-type instructions (which must go in I slots) earlier in the issue group when possible. This way, the later instructions in I slots can be issued to available M ports. Since not all processors support this (such as the Itanium processor), it is preferable to place A-type instructions in M slots.
- Most floating-point load types can be issued to any of the four memory ports, not just M0 and M1. Control speculation-related (advanced and check) and pair floating-point loads are the exceptions which can only be issued to ports M0 and M1. When scheduling a mix of FP loads, advanced FP loads, integer loads, and `lfetch` instructions, ensure that regular FP loads are scheduled late in the issue group so that if necessary, they can be issued to the M2 and M3 ports. This frees the M0 and M1 ports needed by `lfetch` instructions or more restrictive load types.
- Avoid using `nop.f`. It risks unintended stalls due to outstanding long latency instructions. For example, a write to FPSR is a multiple-cycle operation. Any floating-point operation, including a `nop.f`, will stall until the write is completed.
- On the Itanium processor, MFI was a commonly used template to facilitate dual issue. There are many other dual issue template pairs on the Itanium 2 processor so using this template should no longer be necessary.

9.2 Optimal Use of lfetch

The `lfetch` instruction is key to achieving good performance on the Itanium 2 processor in many memory-related situations. `lfetch` allows the L1D to often be a hit for integer data. This has the benefit of allowing the L1D cache to filter requests to the L2. Many L2 conflicts can be avoided by ensuring integer loads hit in the L1D and thus, never are seen by the L2. The fewer requests the L2 sees, the fewer requests conflict.

`lfetch` instructions require careful use. Carelessly placing `lfetch` instructions may lower performance. Refer to Chapter 6, “Memory Subsystem” for details regarding the Itanium 2 processor cache structures. The following guidelines were developed with regard to the memory subsystem:

- The maximum number of outstanding `lfetch` operations to L3 or memory, the sum of both data and instruction requests, may not exceed 16.
- `lfetch` instructions are restricted to only memory ports M0 and M1 while FP loads (not `ldfpd` or `ldfps`) can be issued on any of the four memory ports. Therefore, when mixing `lfetch` instructions with FP loads, `lfetch` instructions should be scheduled early in issue groups. For example, if two FP loads and an `lfetch` are to be scheduled in the same cycle, the `lfetch` should be scheduled in the first bundle so that it will be issued on one of the first two memory ports. If the two FP loads are scheduled first, the hardware will insert an implicit stop before issuing the `lfetch` instruction.
- The Itanium 2 `lfetch.excl` instruction will bring data into the L2 cache in the M state. The `.excl` completer should only be used when the data brought in by the `lfetch` will shortly be modified by store instructions.
- The Itanium 2 `lfetch` instructions will not bring the data into the cache if a DTLB entry providing translation and protection information is not available. To ensure the `lfetch` instruction completes a HPW walk and possibly generates a TLB translation or protection fault, the `.fault` completer should be used. Since there may be high cost associated with these events, the `.fault` completer should not be used for speculative addresses.
- `lfetch` instructions may have effects in the cache hierarchy that make their use high cost. These effects include:
  - — Acquiring L2 resources such as the L2 OzQ.
  - — Arbitration for access to the L2 data arrays and thus becoming a candidate for an L2 bank conflict.
  - — Recirculation of the `lfetch` in the case of a secondary L2 miss.

The effects of the L2 recirculate for a secondary L2 miss can be mitigated by placing `.nt` completers on the `lfetch`. The `.nt` hints keep the `lfetch` from causing an L1D fill and allows the `lfetch` to be removed from the L2 OzQ. However, the non-temporal completer is not absolutely necessary because the L2 OzQ logic can recognize when any `lfetch` instruction is a secondary L2 miss, and does not perform an L1D fill to prevent it from allocating in the L2 OzQ.

In the case where an `lfetch` hits the L2, it takes L2 OzQ resources, causes other request to cancel, and may get canceled itself as if it actually reads the L2 data array regardless of the `.nt` hint or actual need to fill the L1D.

9.3 Data Streaming

There are several methods to handle long, high-bandwidth data streams. This section lists several possible solutions and discusses some of the benefits and costs of each.

9.3.1 Floating-Point Data Streams

Floating-point data resides in the L2 cache. Here, the `lfetch.fault.nt1` instruction should be issued only once per L2 cache line for the source, and the `lfetch.fault.excl.nt1` instruction should be issued only once per L2 cache line for the destination. The `.fault` completer is used to ensure that the data enters into the cache hierarchy, even if it results in an L2 DTLB miss or VHPT miss. The `.nt1` completer ensures that the floating-point data will not displace data residing in the L1D. The `.nt1` completer also allows an `lfetch` instruction that is a secondary L2 miss to avoid allocation in the L2 OzQ. This is important for situations where the design of the data streaming code cannot avoid additional requests to an L2 line without

performance loss. The `.excl` completer for the destination stream will ensure the data is ready to be modified.

When data is accessed as an L2 hit, care should be taken to avoid L2 bank conflicts among request groups. This is necessary to ensure L2 5- and 7-cycle bypasses are available. Latency is not generally a concern for floating-point code, however, in streaming situations, the lifetime of an operation in the L2 OzQ coupled with the size of the OzQ may cause core stalls from the L2 control logic to think the OzQ is full. A lower latency means a shorter lifetime in the OzQ and effectively more OzQ entries are available.

9.3.2 Integer Data Streams

Integer data streams are more complicated than floating-point streams because, in some instances, getting the data into the L1D will be important for performance. Streaming from the L1D presents several problems. First, each load operation hits in the L1D and requires integer register return resources even when it misses the L1D. This makes it difficult for L1D misses to return data to the register file without impacting the flow of new L1D misses. Second, each fill operation will take an additional cycle to complete. Third, the need to fill the L1D eliminates an opportunity for the L2 OzQ to remove secondary L2 miss `lfetch` instructions. This is significant because the L1D line size is half of the L2's and one `lfetch` per L1D line will result in at least one secondary L2 miss access for every L2 line thus limiting L2 OzQ throughput.

One approach would be to use three separate `lfetch` instructions. An `lfetch.fault.nt1` would bring the data into the L2. Later, when the data is in the L2, `lfetch.fault` instructions can hit in the L2 cache and bring the data into the L1D. This makes the `lfetch` instructions asymmetric and requires several load memory slots.

An optimization to the three `lfetch` approach above would use only two separate `lfetch.fault` instruction, but stage them such that the first will bring data into L2 and the L1D. Then, when the L2 is filled from the first request, the second `lfetch` can bring the data into the L1D without being a secondary L2 miss (the L2 is filled so the `lfetch` is an L2 hit). This frees an additional load memory slot and makes the `lfetch` instructions re-usable.

An outstanding L1D fill may be invalidated by a store to the same line. Using `lfetch` instructions for even small data streams can result in a significant performance increase provided the `lfetch` fills the L1D before the store to the line is seen.

Also, since all loads that hit in the L1D never allocate into the L2 OzQ, using `lfetch` instructions to ensure an L1D hit may also help performance by limiting L2 OzQ to only store data and `lfetch` requests. This relieves pressure on the limited OzQ resources and reduces the possibility of conflicts among OzQ entries.

9.3.3 Store Data Streams

Since store instructions are always seen by the L2, there is no benefit to bringing store destination data into the L1D. There are many benefits to using an `lfetch.fault.excl.nt1` completer for destination streams. For instance, the `.nt1` hint allows secondary L2 misses to be removed and the core is not slowed by the L1D fills. Also, the `.excl` hint ensures that the L2 data is ready to receive the store data.

9.4 Control and Data Speculation

The Itanium 2 processor reduces the costs associated with control and data speculation in the ALAT via fast deferral and low latency fix up. As such, additional performance may be realized by tuning the code generation to aggressively use speculation. Some speculation considerations are specific to the Itanium processor and do not apply to the Itanium 2 processor. If speculation is more aggressive, then more calls to fix up code will be encountered. For the Itanium processor, the fix up code was often moved to cold pages very far from the actual speculation. The heuristic for placing fix up code near or far from the point of speculation should be revisited and include profile information in the decision matrix.

9.5 Known L2 Miss Bundle Placement

Given the Itanium 2 processor design, it is slightly better to put instructions which are known to miss the L2 cache on memory port 0 (allocate the first memory op in the issue group). This will allow, when possible, a speculative request to be made to L3. If the memory request that needs to go to L2 is in M1, M2, or M3, then they will need to wait until they can be reissued out of the L2 OzQ.

9.6 Avoid Known L2 Cancel and Recirculate Conditions

The most predictable L2 cancel is an L2 bank conflict. These can be avoided by carefully organizing L2 accesses or by bringing the data into the L1D with an `lfetch` instruction and avoiding the L2 entirely.

The most predictable L2 recirculate is for secondary L2 miss accesses. These can be avoided by using the `lfetch` instruction to bring data into the L2. Only `lfetch` instructions that do not fill L1D are not counted as a secondary access. If an `lfetch` is the primary L2 miss and a load is the secondary L2 miss, then the load will still need to recirculate, as it must eventually return data to the core. It is important to schedule L2 miss `lfetch` instructions far in front of the load to avoid this situation.

9.7 Instruction Bundling

The Itanium 2 processor can completely issue almost all bundle template combinations. Provided the ILP is available, closing the correct bundling and instruction scheduling may benefit performance. There are two concerns here. First, place more restrictive instructions early in the issue group and, where possible, transform restrictive instructions. The simple instruction `nop.i` must issue to an I port, however, an add can issue on either an M or I port. The `nop.i` should be scheduled early to ensure it receives its needed I port. An alternative would be to replace the `nop.i` with an instruction that is effectively a `nop` (such as `add r3=r0, r3`) which can issue on either an I or M port.

9.8 Branches

The following branch and branch prediction related optimization suggestions are covered in detail in Chapter 7, "Branch Instructions and Branch Prediction." They are summarized here.

9.8.1 Single Cycle Branches

The Itanium 2 processor cannot support single cycle loop branches without some penalty in some iterations of the loop. Unroll the loop to at least two cycles to get expected performance. This may come at a small cost to code size.

9.8.2 Perfect Loop Prediction

Also, perfect loop prediction only predicts the final iteration of the loop. As such, the Itanium 2 processor considers the branch hints in predicting the branches. The Itanium 2 processor requires `ar.ec` to be set correctly (i.e. if there is no epilogue, set `ar.ec`=0 not to 1 as the Itanium processor expected).

9.8.3 Branch Targets

Branch targets should be aligned on 32-byte boundaries to ensure that the front-end can deliver two bundles per cycle to the back-end.

Performance Monitoring 10

10.1 Introduction

This chapter defines the performance monitoring features of the Itanium 2 processor. The Itanium 2 processor provides four 48-bit performance counters, 100+ monitorable events, and several advanced monitoring capabilities. This chapter outlines the targeted performance monitor usage models, defines the software interface and programming model, and lists the set of monitored events.

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 can 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. On the Itanium 2 processor, we expect to measure the behavior of real-world Itanium architecture-based applications and operating systems as well as mixed IA-32 and Itanium architecture-based code. 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 the document is split into the following sections:

- Section 10.2, “Performance Monitor Programming Models” discusses how performance monitors are used, and presents various Itanium 2 processor performance monitoring programming models.
- Section 10.3, “Performance Monitor State” defines the Itanium 2 processor specific PMC/PMD performance monitoring registers.
- Chapter 11, “Performance Monitor Events” gives an overview of the Itanium 2 processor event list.

10.2 Performance Monitor Programming Models

This section introduces the Itanium 2 processor performance monitoring features from a programming model point of view and describes how the different event monitoring mechanisms can be used effectively. The Itanium 2 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 10.2.1, “Workload Characterization” discusses the Itanium 2 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 Itanium architectural features such as predication

and speculation benefit from run-time profile information to optimize instruction schedules. The Itanium 2 processor supports instruction level statistical profiling of branch mispredicts and cache misses. Details of the Itanium 2 processor's profiling support are described in Section 10.2.2, "Profiling."

10.2.1 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, 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 10.2.1.1, "Event Rate Monitoring" 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 10.2.1.4, "Cycle Accounting" discusses cycle accounting.

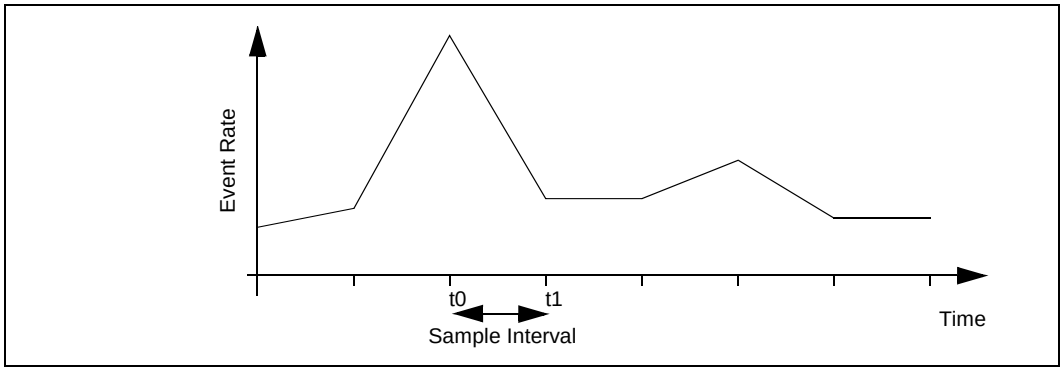
10.2.1.1 Event Rate Monitoring

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 Itanium 2 processor events that count the number of retired Itanium instructions (IA64_INST_RETIRED.u) and the number of elapsed clock cycles (CPU_CYCLES) allow a workload's instructions per cycle (IPC) to be computed as follows:

- IPC = (IA64_INST_RETIRED.u_{t1} - IA64_INST_RETIRED.u_{t0}) / (CPU_CYCLES_{t1} - CPU_CYCLES_{t0})

Time-based sampling is the basis for many performance debugging tools [VTune™, gprof, WinNT]. As shown in Figure 10-1, 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 10-1. Time-Based Sampling



On the Itanium processor, many event types, e.g. TLB misses or branch mispredicts are limited to a rate of one per clock cycle. These are referred to as "single occurrence" events. However, in the Itanium 2 processor, multiple events of the same type may occur in the same clock. We refer to such events as "multi-occurrence" events. An example of a multi-occurrence events on the

Itanium 2 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. The next two sections describe the basic Itanium 2 processor mechanisms for monitoring single and multi-occurrence events.

10.2.1.2 Single Occurrence Events and Duration Counts

A single occurrence event can be monitored by any of the Itanium 2 processor performance counters. For all single occurrence events, a counter is incremented by up to one per clock cycle. Duration counters that count the number of clock cycles during which a condition persists are considered “single occurrence” events. Examples of single occurrence events on the Itanium 2 processor are TLB misses, branch mispredictions, and cycle-based metrics.

10.2.1.3 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 Itanium 2 processor are retired instructions or the number of live entries in the memory request queue.

Thresholding capabilities are available in the Itanium 2 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 10-1 in which the average outstanding requests/cycle = 15/8 = 1.825, and the average latency per request = 15/5 = 3 cycles.

Table 10-1. Average Latency per Request and Requests per Cycle Calculation Example

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

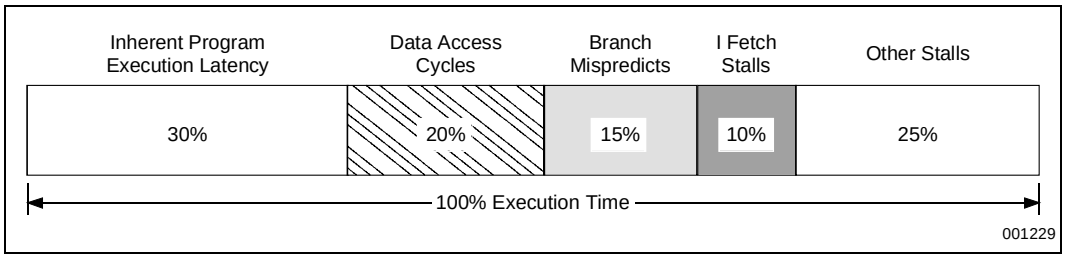
The Itanium 2 processor provides the following capabilities to support event rate monitoring:

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

10.2.1.4 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 IPC rate. If a low IPC occurs concurrently with a peak of cache miss activity, chances are that cache misses are causing a performance problem. To eliminate such guess work, the Itanium 2 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 10-2, 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 10-2. Itanium® Processor Family Cycle Accounting



The Itanium 2 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 11.6, "Stall Events."

10.2.2 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 Itanium architectural features such as predication and speculation.

To support profiling, performance monitor counts have to be associated with program locations. The following mechanisms are supported directly by the Itanium 2 processor's performance monitors:

- Program Counter Sampling
- Miss Event Address Sampling: Itanium 2 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 the next three subsections.

10.2.2.1 Program Counter Sampling

Application tuning tools like [VTune, gprof] use time-based or event-based sampling of the program counter and other event counters to identify performance critical functions and basic blocks. As shown in Figure 10-3, the sampled points can be histogrammed by instruction addresses. For application tuning, statistical sampling techniques have been very successful, because the programmer can rapidly identify code hot spots in which the program spends a significant fraction of its time, or where certain event counts are high.

Figure 10-3. Event Histogram by Program Counter

Program counter sampling points the performance analysts at code hot spots, but does not indicate what caused the performance problem. Inspection and manual analysis of the hot-spot region along with a fair amount of guess work are required to identify the root cause of the performance problem. On the Itanium 2 processor, the cycle accounting mechanism (described in Section 10.2.1.4, "Cycle Accounting") can be used to directly measure an application's microarchitectural behavior.

The Itanium architectural interval timer facilities (ITC and ITM registers) can be used for time-based program counter sampling. Event-based program counter sampling is supported by a dedicated performance monitor overflow interrupt mechanism described in detail in 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*.

To support program counter sampling, the Itanium 2 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

10.2.2.2 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 an Pentium® Pro processor, this distance increases to approximately 32 dynamic instructions. On the Itanium 2 processor, it is approximately 48 dynamic instructions. If program counter sampling is used for miss event address identification on the Itanium 2 processor, a miss event might be associated with an instruction almost five dynamic basic blocks away from where it actually occurred (assuming that 10% of all instructions are branches). Therefore, it is essential for hardware to precisely identify an event’s address.

The Itanium 2 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 four deep *branch trace buffer* captures sequences of branch instructions. Table 10-2 summarizes the capabilities offered by the Itanium 2 processor EARs and the branch trace buffer. Exposing miss event addresses to software allows them to be monitored either by sampling or by code instrumentation. This eliminates the need for trace generation to identify and solve performance problems and enables performance analysis by a much larger audience on unmodified hardware.

Table 10-2. Itanium® 2 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 Who 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.

Table 10-2. Itanium® 2 Processor EARs and Branch Trace Buffer (Continued)

Event Address Register	Triggers On	What is Recorded
Data TLB (DTLB)	Data references that miss L1 DTLB	Instruction Address Data Address Who serviced L1 DTLB miss: L2 DTLB, VHPT or software
Branch Trace Buffer	Branch Outcomes	Branch Instruction Address Branch Target Instruction Address Mispredict status and reason

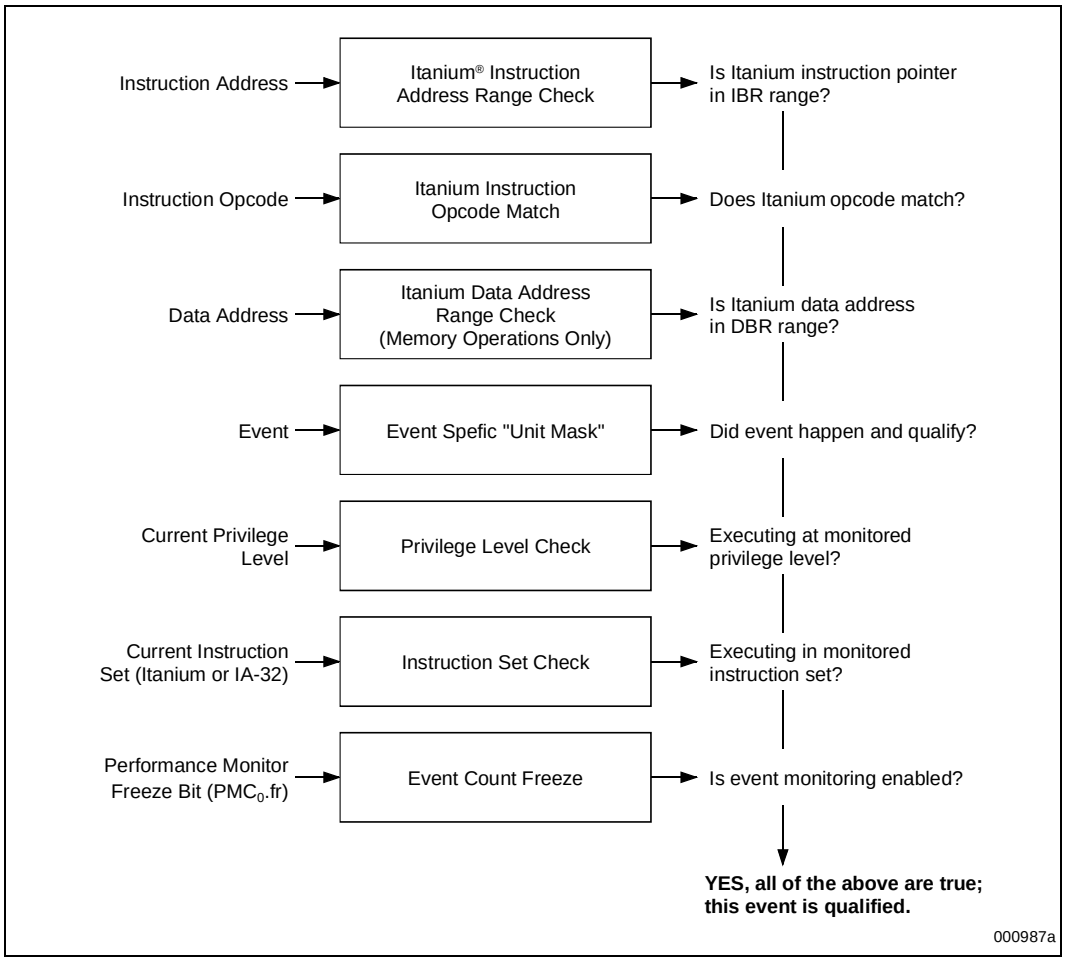
The Itanium 2 processor EARs enable statistical sampling by configuring a performance counter to count, for instance, the number of data cache misses or retired instructions. The performance counter value is set up to interrupt the processor after a predetermined number of events have been observed. 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.

10.2.3 Event Qualification

On the Itanium 2 processor, performance monitoring can be confined to a subset of all events. As shown in Figure 10-4 events can be qualified for monitoring based on an instruction address range, a particular instruction opcode, a data address range, an event-specific “unit mask” (umask), the privilege level and instruction set the event was caused by, and the status of the performance monitoring freeze bit (PMC_0.fr).

- Itanium Instruction Address Range Check: The Itanium 2 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-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 Itanium 2 processor’s instruction address range check operates only during Itanium-based code execution, i.e. when `PSR.is` is zero. For details, see Itanium Opcode Match and Address Range Check Registers ($PMC_{8,9}$).

Figure 10-4. Itanium® 2 Processor Event Qualification



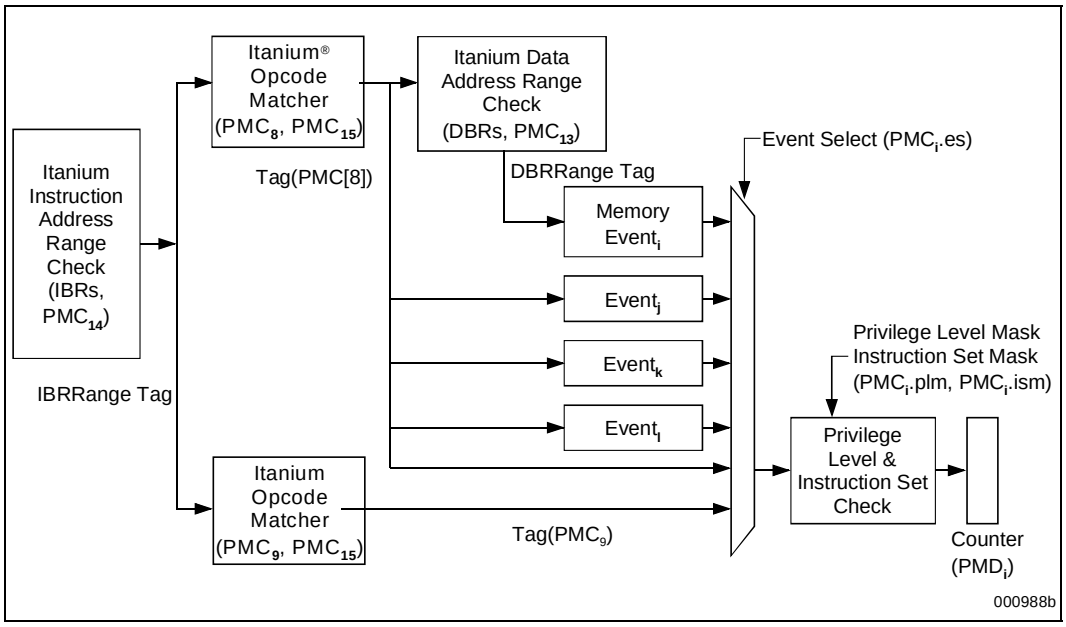
- Itanium Instruction Opcode Match: The Itanium 2 processor provides two independent Itanium opcode match registers each of which match the currently issued instruction encodings with a programmable opcode match and mask function. The resulting match events can be selected as an event type for counting by the performance counters. This allows histogramming of instruction types, usage of destination and predicate registers as well as basic block profiling (through insertion of tagged NOPs). The opcode matcher operates only during Itanium-based code execution, i.e. when `PSR.is` is zero. Details are described in Section 10.3.4.
- Itanium Data Address Range Check: The Itanium 2 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 10.3.6.
- 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 Itanium 2 processor, events from the branch, memory and bus units support a variety of unit masks. For details, refer to the event pages in Chapter 11, “Performance Monitor Events.”

- Privilege Level: Two bits in the processor status register are provided to enable selective process-based event monitoring. The Itanium 2 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 10.3.1, "Performance Monitor Control and Accessibility."
- Instruction Set: The Itanium 2 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 and IA-32 contributions. For details, refer to Section 10.3.1, "Performance Monitor Control and Accessibility.".
- Performance Monitor Freeze: Event counter overflows or software can freeze event monitoring. When frozen, no event monitoring takes place until software clears the monitoring freeze bit (PMC_0.fr). This ensures that the performance monitoring routines themselves, e.g. counter overflow interrupt handlers or performance monitoring context switch routines, do not "pollute" the event counts of the system under observation. For details refer to Section 7.2.4 of Volume 2 of the *Intel® Itanium™ Architecture Software Developer's Manual*.

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

The Itanium 2 processor allows various event qualification mechanisms to be combined by providing the instruction tagging mechanism shown in Figure 10-5.

Figure 10-5. Instruction Tagging Mechanism in the Itanium® 2 Processor



During Itanium instruction execution (`PSR.is` is zero), the instruction address range check is applied first. The resulting address range check tag (IBRRangeTag) is passed to two opcode matchers that combine the instruction address range check with the opcode match. Each of the two combined tags (Tag(PMC_8) and Tag(PMC_9)) can be counted as a retired instruction count event (for details refer to event description "IA64_TAGGED_INST_RETIRED" on page 11-52).

One of the combined Itanium address range and opcode match tags, Tag(PMC_8), 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 DBRRangeTag).

As summarized in Table 10-3, data address range checking can be combined with opcode matching and instruction range checking on the Itanium 2 processor. Additional event qualifications based on the current privilege level and the current instruction set can be applied to all events and are discussed in Section 10.2.3.2, "Privilege Level Constraints" and Section 10.2.3.3, "Instruction Set Constraints."

Table 10-3. Itanium® 2 Processor Event Qualification Modes

Event Qualification Modes	Opcode Match Enable PMC_{15}.ibrp0-pmc8	Opcode Matching PMC_8	Instruction Address Range Check Enable PMC_{14}.ibrp0	Data Address Range Check [PMC_{13}.enable-dbrp# PMC_{13}.dbrp#] (mem pipe events only)
Unconstrained Monitoring (all events)	x	0xffff_ffff_ffff_ffff	x	[1,11] or [0,xx]
Instruction Address Range Check Only	x	0xffff_ffff_ffff_fffe	0	[1,00]
Opcode Matching Only	1	Desired Opcodes	x	[1,01]
Data Address Range Check Only	x	0xffff_ffff_ffff_ffff	x	[1,10]
Instruction Address Range Check and Opcode Matching	1	Desired Opcodes	0	[1,01]
Instruction and Data Address Range Check	x	0xffff_ffff_ffff_fffe	0	[1,00]
Opcode Matching and Data Address Range Check	1	Desired Opcodes	x	[1,00]

10.2.3.2 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 Itanium 2 processor performance monitors control is discussed in Section 10.3.1, "Performance Monitor Control and Accessibility.".

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 10-6, Figure 10-7, and Figure 10-8, 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 performance monitoring chapter of the *Intel® Itanium® Architecture Software Developer's Manual*.

10.2.3.3 Instruction Set Constraints

On the Itanium 2 processor, monitoring can additionally be constrained based on the currently executing instruction set as defined by `PSR.is`. This capability is supported by the four generic performance counters, as well as the opcode matching and instruction and data event address registers. However, the branch trace buffer only supports Itanium-based code execution. When Itanium architecture only features are used, the corresponding PMC register instruction set mask (PMC_i.ism) should be set to Itanium architecture only (01) to ensure that events generated by IA-32 code do not corrupt the Itanium 2 processor event counts.

Figure 10-6. Single Process Monitor

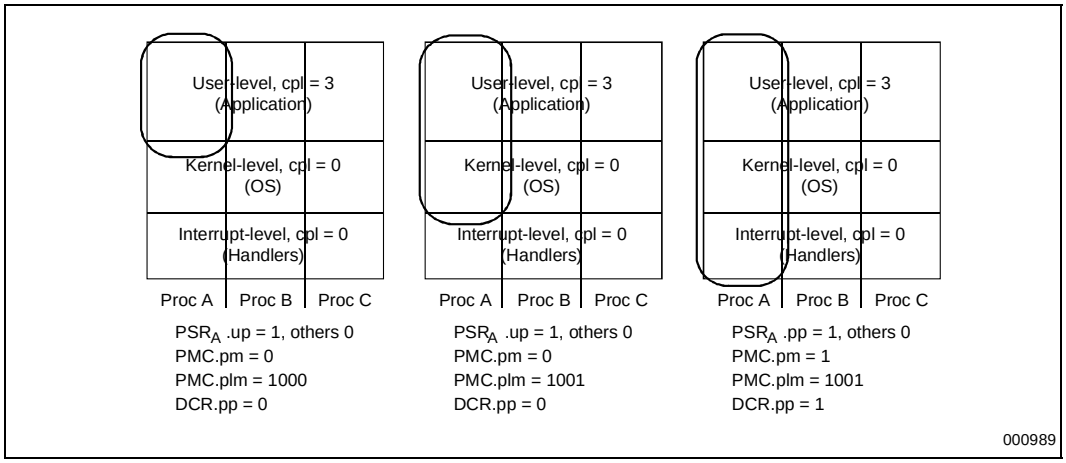


Figure 10-7. Multiple Process Monitor

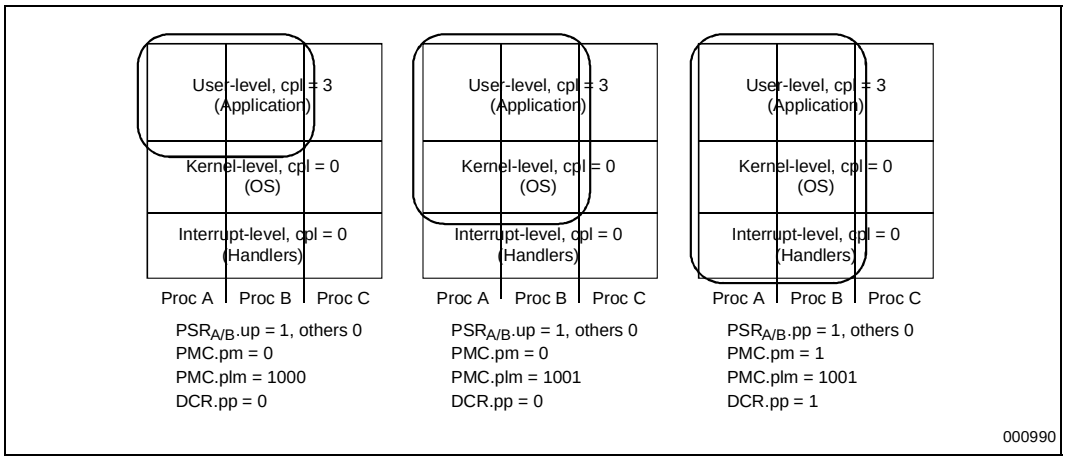
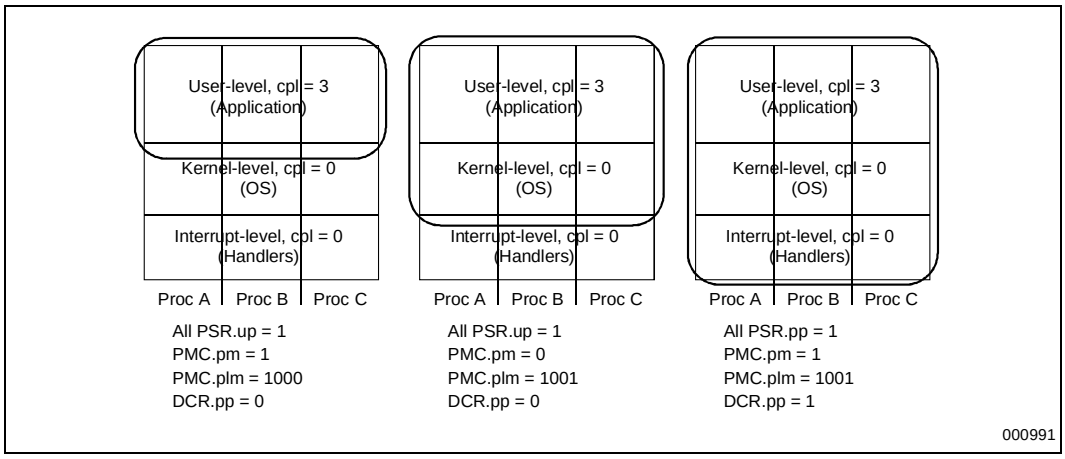


Figure 10-8. System Wide Monitor



10.2.4 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.

10.3 Performance Monitor State

Two sets of performance monitor registers are defined. Performance Monitor Configuration (PMC) registers are used to configure the monitors. Performance Monitor Data (PMD) registers provide data values from the monitors. This section describes the Itanium 2 processor performance monitoring registers which expands on the Itanium architectural definition. As shown in Figure 10-9 the Itanium 2 processor provides four 48-bit performance counters ($PMC/PMD_{4,5,6,7}$ pairs), and the following model-specific monitoring registers: instruction and data event address registers (EARs) for monitoring cache and TLB misses, a branch trace buffer, two opcode match registers, and an instruction address range check register.

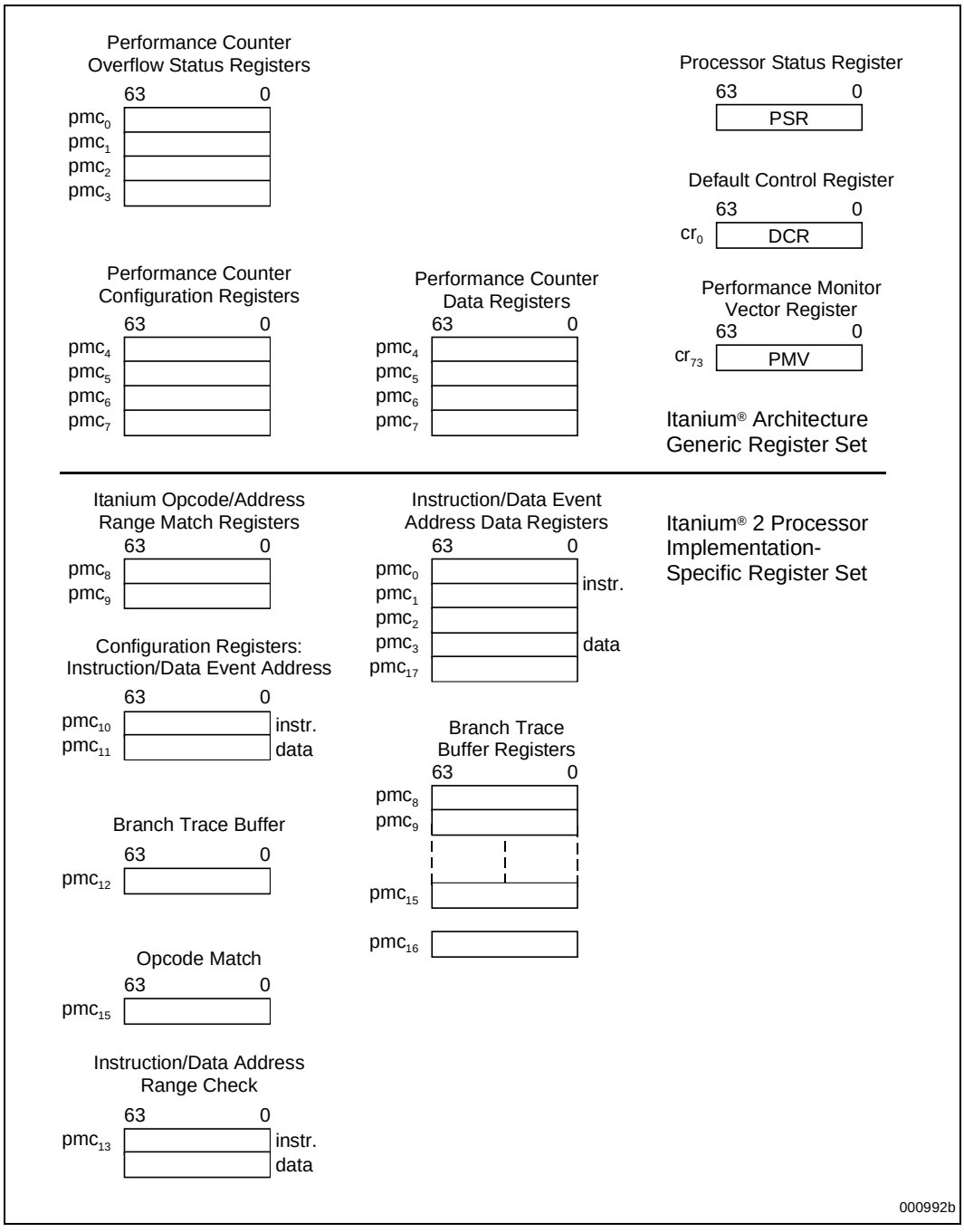
Table 10-4 defines the PMC/PMD register assignments for each monitoring feature. The interrupt status registers are mapped to $PMC_{0,1,2,3}$. The four generic performance counter pairs are assigned to $PMC/PMD_{4,5,6,7}$. The event address registers and the branch trace buffer are controlled by three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

($PMC_{8,9}$) and an opcode match configuration register (PMC_{15}) 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_{14}).

Table 10-4. Itanium® 2 Processor Performance Monitor Register Set

Monitoring Feature	Configuration Registers (PMC)	Data Registers (PMD)	Description
Interrupt Status	$PMC_{0,1,2,3}$	none	See Section 10.3.3, “Performance Monitor Overflow Status Registers (PMC0,1,2,3)”
Event Counters	$PMC_{4,5,6,7}$	$PMD_{4,5,6,7}$	See Section 10.3.2, “Performance Counter Registers”
Opcode Matching	$PMC_{8,9,15}$	none	See Section 10.3.4, “Opcode Match Check (PMC8,9,15)”
Instruction EAR	PMC_{10}	$PMD_{0,1}$	See Section 10.3.7.1, “Instruction EAR (PMC10, PMD0,1)”
Data EAR	PMC_{11}	$PMD_{2,3,17}$	See Section 10.3.8, “Data EAR (PMC11, PMD2,3,17)”
Branch Trace Buffer	PMC_{12}	$PMD_{8\text{-}16}$	See Section 10.3.9.2, “Branch Trace Buffer Reading”
Instruction Address Range Check	PMC_{14}	none	See Section 10.3.5, “Instruction Address Range Matching”
Memory Pipeline Event Constraints	PMC_{13}	none	See Section 10.3.6, “Data Address Range Matching (PMC13)”

Figure 10-9. Itanium® 2 Processor Performance Monitor Register Mode



10.3.1 Performance Monitor Control and Accessibility

In order to use performance monitor features, the power to the PMU should be turned on by setting PMC_4.enable to 1. At reset, this bit will be set. To provide power savings, this bit can be cleared to turn off the clocks to all PMDs, PMCs (with the exception of PMC_4), and other non-critical circuitry.

Once the power is turned on, 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). Instruction set masking based on PMC_i.ism is a Itanium 2 processor model-specific feature. Event collection for a monitor is enabled under the following constraints on the Itanium 2 processor:

```
Monitor Enable_i =(not PMC_0.fr) and PMC_i.plm[PSR.cpl] and ((not PMC_i.ism[PSR.is]) or
(PMC_i=12)) and ((not (PMC_i.pm) and PSR.up) or (PMC_i.pm and PSR.pp))
```

Figure 10-10 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 10-5 defines per monitor controls that apply to $PMC_{4,5,6,7,10,11,12}$. As defined in Table 10-4, "Itanium® 2 Processor Performance Monitor Register Set," each of these PMC registers controls the behavior of its associated performance monitor data registers (PMD). The Itanium 2 processor model-specific PMD registers associated with instruction/data EARs and the branch trace buffer ($PMD_{0,1,2,3,8-17}$) can be read only when event monitoring is frozen (PMC_0.fr is one).

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

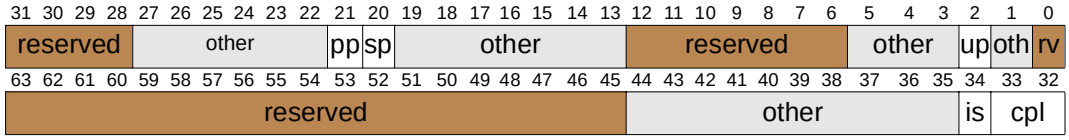


Table 10-5. Performance Monitor PMC Register Control Fields ($PMC_{4,5,6,7,\ 0,11,12}$)

Field	Bits	Description
plm	3:0	Privilege Level Mask - controls performance monitor operation for a specific privilege level. Each bit corresponds to one of the 4 privilege levels, with bit 0 corresponding to privilege level 0, bit 1 with privilege level 1, etc. A bit value of 1 indicates that the monitor is enabled at that privilege level. Writing zeros to all plm bits effectively disables the monitor. In this state, the Itanium 2 processor will not preserve the value of the corresponding PMD register(s).

Table 10-5. Performance Monitor PMC Register Control Fields ($PMC_{4,5,6,7,\ 0,11,12}$) (Continued)

Field	Bits	Description
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_{10} this field is implemented in bit [4].
ism	25:24	Instruction Set Mask - controls performance monitor operation based on the current instruction set. The instruction set mask applies to $PMC_{4,5,6,7,10,11}$ but not to PMC_{12}. 00: monitoring enabled during Itanium and IA-32 instruction execution (regardless of PSR.is) 10: bit 24 low enables monitoring during Itanium instruction execution (when PSR.is is zero) 01: bit 25 low enables monitoring during IA-32 instruction execution (when PSR.is is one) 11: disables monitoring NOTE: In PMC_{10} this is implemented in [15:14]. PMC_{12} does not have this field.

10.3.2 Performance Counter Registers

The Itanium 2 processor provides four generic performance counters ($PMC/PMD_{4,5,6,7}$ pairs). The implemented counter width on the Itanium 2 processor is 48 bits ([47] indicates overflow condition). More than the Itanium processor, PMC/PMD pairs on the Itanium 2 processor are symmetrical, i.e. nearly all event types can be monitored by all counters. There are exceptions within some of the cache counters. See Section 11.8.2, "L1 Data Cache Events" and Section 11.8.3, "L2 Unified Cache Events" for more information. These counters can track events whose maximum per-cycle event increment is up to seven.

Figure 10-11 and Table 10-6 define the layout of the Itanium 2 processor Performance Counter Configuration Registers ($PMC_{4,5,6,7}$). 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`) and unit mask (`umask`) 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 10-11. Itanium® 2 Processor Generic PMC Registers ($PMC_{4,5,6,7}$)

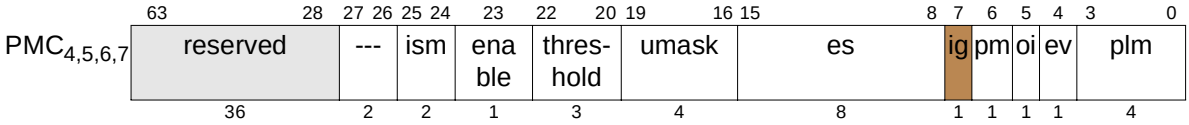


Table 10-6. Itanium® 2 Processor Generic PMC Register Fields ($PMC_{4,5,6,7}$)

Field	Bits	Description
plm	3:0	Privilege Level Mask. See Table 10-5 "Performance Monitor PMC Register Control Fields (PMC4,5,6,7, 0,11,12)."
ev	4	External visibility - When 1, an external notification (BPM pin strobe) is provided whenever the counter overflows. External notification occurs regardless of the setting of the oi bit (see below). On the Itanium 2 processor, PMC_4 external notification strobes the BPM0 pin, PMC_5 strobes the BPM1 pin, PMC_6 strobes the BPM2 pin, and PMC_7 strobes the BPM3 pin.

Table 10-6. Itanium® 2 Processor Generic PMC Register Fields ($PMC_{4,5,6,7}$) (Continued)

Field	Bits	Description
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 10-5 “Performance Monitor PMC Register Control Fields (PMC4,5,6,7, 0,11,12).”
ig	7	reserved
es	15:8	Event select - selects the performance event to be monitored. Itanium 2 processor event encodings are defined in Chapter 11, “Performance Monitor Events.”
umask	19:16	Unit Mask - event specific mask bits (see event definition for details)
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.
enable	23	PMC_4 Only. Enables use of the PMUs. A 1 must be written for the PMUs to function. Power up value is 1.
ism	25:24	Instruction Set Mask. See Table 10-5 “Performance Monitor PMC Register Control Fields (PMC4,5,6,7, 0,11,12).”
---	27:26	Must write 0 for proper PMU operation.
ignored	63:28	Read zero, Writes ignored.

Figure 10-12 and Table 10-7 defines the layout of the Itanium 2 processor Performance Counter Data Registers ($PMD_{4,5,6,7}$). 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 $PMD_{4,5,6,7}$ without disabling event collection. Any read of the PMD from software without the appropriate privilege level will return 0 (See “plm” in Table 10-6). The processor ensures that software will see monotonically increasing counter values.

Figure 10-12. Itanium® 2 Processor Generic PMD Registers ($PMD_{4,5,6,7}$)

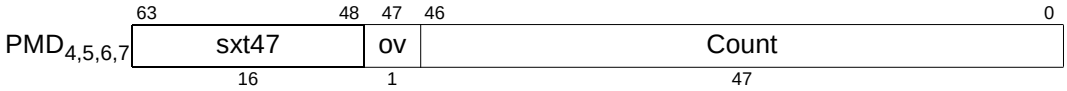


Table 10-7. Itanium® 2 Processor Generic PMD Register Fields

Field	Bits	Description
sxt47	63:48	Writes are ignored, Reads return the value of bit 47, so count values appear as sign extended.
ov	47	Overflow bit (carry out from bit 46). NOTE: Writes to initialize the PMD should write 0 to this bit.
count	46:0	Event Count. The counter is defined to overflow when the count field wraps (carry out from bit 46).

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

As previously mentioned, the Itanium 2 processor supports four performance monitoring counters. The overflow status of these four counters is indicated in register PMC_0. As shown in Figure 10-13 and Table 10-8 only PMC_0[7:4,0] bits are populated. All other overflow bits are ignored, i.e. they read as zero and ignore writes.

Figure 10-13. Itanium® 2 Processor Performance Monitor Overflow Status Registers ($PMC_{0,1,2,3}$)

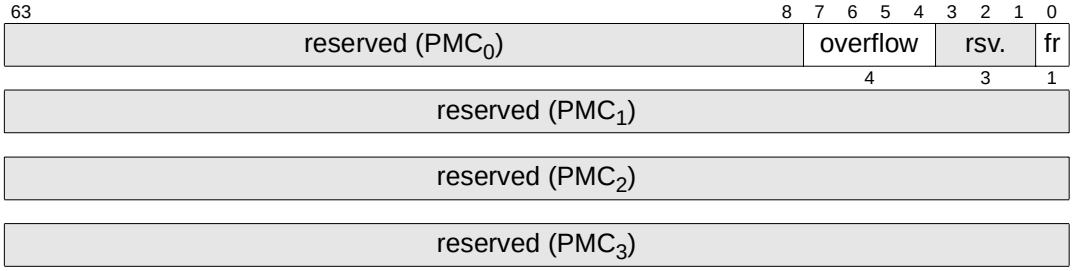


Table 10-8. Itanium® 2 Processor Performance Monitor Overflow Register Fields ($PMC_{0,1,2,3}$)

Register	Field	Bits	Description
PMC_0	fr	0	Performance Monitor "freeze" bit - When 1, event monitoring is disabled. When 0, event monitoring is enabled. This bit is set by hardware whenever a performance monitor overflow occurs and its corresponding overflow interrupt bit (PMC.oi) is set to one. SW is responsible for clearing it. When the PMC.oi bit is not set, then counter overflows do not set this bit.
PMC_0	ignored	3:1	Read zero, Writes ignored.
PMC_0	overflow	7:4	Event Counter Overflow - When bit n is one, indicate that the 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	ignored	63:8	Read zero, Writes ignored.
$PMC_{1,2,3}$	ignored	63:0	Read zero, Writes ignored.

10.3.4 Opcode Match Check ($PMC_{8,9,15}$)

The Itanium 2 processor allows event monitoring to be constrained based on the instruction address and/or Itanium encoding (opcode) of an instruction. Registers PMC_{15} and PMC_{14} (Section 10.3.5, "Instruction Address Range Matching") are used to enable these features. Registers $PMC_{8,9}$ allow configuring these features. For memory related events, the appropriate bits must be set in PMC_{13} to enable this feature. Please refer to Section 10.3.6, "Data Address Range Matching (PMC13)" for details. Unlike in the Itanium processor, the opcode matcher in the Itanium 2 processor operates during both Itanium-based and IA-32 code execution. When operating in IA-32 mode it checks for Itanium opcodes.

Figure 10-14 and Table 10-9 describe the fields of $PMC_{8,9}$ registers. Figure 10-15 and Table 10-10 describes the register PMC_{15}. All combinations of bits [63:60] are supported. To match A-slot instruction, set bits [63:62] to 11. To match all instruction types, bits [63:60] should be set to 1111. To ensure that all events are counted independent of the opcode matcher, all mifb and all mask bits of $PMC_{8,9}$ should be set to one (all opcodes match).

PMC_9 only qualifies the event IA64_TAGGED_INST_RETIRED. The Itanium 2 processor's opcode constraint for IA64_TAGGED_INST_RETIRED event ANDs PMC_9 results with $IBRP_1$ and $IBRP_3$ matches and PMC_8 results with IBR_0 and $IBRP_2$ matches. PMC_8, however, constrains other downstream events as well. To ensure that all events are counted independent of the opcode matcher, bit[63:60] and bit [29:3] should be set to all ones.

Figure 10-14. Opcode Match Registers ($PMC_{8,9}$)

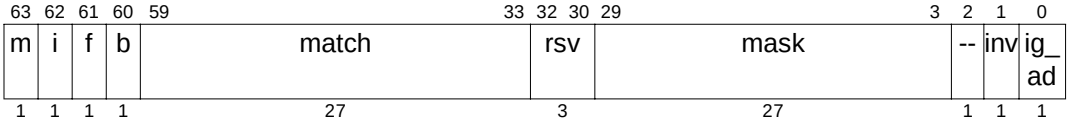


Table 10-9. Opcode Match Register Fields ($PMC_{8,9}$)

Field	Bits	Width	Description
ig_ad	0	1	Ignore Instruction Address Range Checking. 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_9.
inv	1	1	Invert Range Check. If set to 1, the address ranged specified by IBR0-1 is inverted. Effective only when ig_ad bit is set to 0. NOTE: This bit is ignored in PMC_9.
--	2	1	Must write 1 for proper PMU operation.
mask	29:3	27	Bits that mask Itanium® instruction encoding bits [15:3] mask bits for opcode bits[12:0] [29:16] mask bits for opcode bits[40:27] If mask bit is set to 1, the corresponding opcode bit is not used for opcode matching
rsv	32:30	3	Reserved bits
match	59:33	27	Opcode bits against which Itanium instruction encoding to be matched [45:33]: match bits for opcode bits[12:0] [59:46]: match bits for opcode bits[40:27]
b	60	1	If 1: match if opcode is an B-slot
f	61	1	If 1: match if opcode is an F-slot
i	62	1	If 1: match if opcode is an I-slot
m	63	1	If 1: match if opcode is an M-slot

Figure 10-15. Opcode Match Configuration Register (PMC_{15})

Table 10-10. Opcode Match Configuration Register Fields (PMC_{15})

Field	Bits	Description
ibrp0-pmc8	0	1: PMU events will not be constrained by opcode 0: PMU events (including IA64_TAGGED_INST_RETIRED.00) will be opcode constrained by PMC_8
ibrp1-pmc9	1	1: IA64_TAGGED_INST_RETIRED.01 won't be constrained by opcode 0: IA64_TAGGED_INST_RETIRED.01 will be opcode constrained by PMC_9
ibrp2-pmc8	2	1: IA64_TAGGED_INST_RETIRED.10 won't be constrained by opcode 0: IA64_TAGGED_INST_RETIRED.10 will be opcode constrained by PMC_8
ibrp3-pmc9	3	1: IA64_TAGGED_INST_RETIRED.11 won't be constrained by opcode 0: IA64_TAGGED_INST_RETIRED.11 will be opcode constrained by PMC_9

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 42-bit encoding "enco{41:0}" defined the *Intel® Itanium® Architecture Software Developer's Manual*. Each instruction is evaluated against each opcode match register ($PMC_{8,9}$) as follows:

```
Match(PMC_i) = (imatch(itype, PMC_i.mifb) AND ematch(enco,PMC_i.match,PMC_i.mask))
```

Where:

```
imatch(itype,PMC[i].mifb) = (itype=M AND PMC[i].m) OR (itype=I AND PMC[i].i) OR
(itype=F AND PMC[i].f) OR (itype=B AND PMC[i].b)
ematch(enco,match,mask) = AND b=40..27 ((enco{b}=match{b-14}) OR mask{b-14}) AND
b=12..0 ((enco{b}=match{b}) OR mask{b})
```

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

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 10.3.5, "Instruction Address Range Matching") as follows:

```
Tag(PMC_8) = Match(PMC_8) and IBRRangeTag
Tag(PMC_9) = Match(PMC_9) and IBRRangeTag
```

As shown in Figure 10-5 the two tags, Tag(PMC_8) and Tag(PMC_9), are staged down the processor pipeline until instruction retirement and can be selected as a retired instruction count event (see event description "IA64_TAGGED_INST_RETIRED" on page 11-52). In this way, a performance counter ($PMC/PMD_{4,5,6,7}$) can be used to count the number of retired instructions within the programmed range that match the specified opcodes.

The opcodes dispersed to different pipelines are compared to PMC_8; the opcode match is further qualified by a number of user configurable bits (please refer to definition of PMC_{15} in this document) and ANDed with IBRP0 match before being distributed to different places.

Note: Register PMC_{15} must contain the predetermined value of 0xfffffff0. If software modifies any bits not listed in Table 10-10 processor behavior is not defined.

10.3.5 Instruction Address Range Matching

The Itanium 2 processor allows event monitoring to be constrained to a range of instruction addresses. The four architectural Instruction Breakpoint Register Pairs $IBRP_{0-3}$ (IBR_{0-17}) can be used to specify the desired address range. Once programmed this restriction would be applied to all events. In the Itanium 2 processor, registers $PMC_{8,14}$ specify how the resulting address match is applied to the performance monitors. With the exception of IA64_INST_RETIRED and prefetch events, $IBRP_0$ is the only IBR pair used and will be considered the default for this section. For memory related events, the appropriate bits must be set in PMC_{13} to enable this feature. Please refer to Section 10.3.6, "Data Address Range Matching (PMC13)" for details.

Figure 10-16 and Table 10-12 describe the fields of register PMC_{14}. Instruction address range checking is controlled by the "ignore address range check" bit (PMC_8.ig_ad and PMC_{14}.ibrp0). When PMC_8.ig_ad is one (or PMC_{14}.ibrp0 is one), all instructions are tagged regardless of IBR settings. In this mode, events from both IA-32 and Itanium-based code execution contribute to the event count. When both PMC_8.ig_ad and PMC_{14}.ibrp0 are zero, the instruction address range check based on the IBR settings is applied to all Itanium 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 10-11 defines the behavior of the instruction address range checker for different combinations of `PSR.is` and PMC_8.ig_ad or PMC_{14}.ibrp0.

Table 10-11. Itanium® 2 Processor Instruction Address Range Check by Instruction Set

	PSR.is	
PMC_8.ig_ad OR PMC_{14}.ibrp0	0 (Itanium®)	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 Itanium 2 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_8 and further qualified with more user definable bits (see Table 10-12) before being distributed to different places. The events which occur after instruction dispersal stage, will use this new qualified match (ibrp0-pmc8 match).

Figure 10-16. Instruction Address Range Configuration Register (PMC_{14})

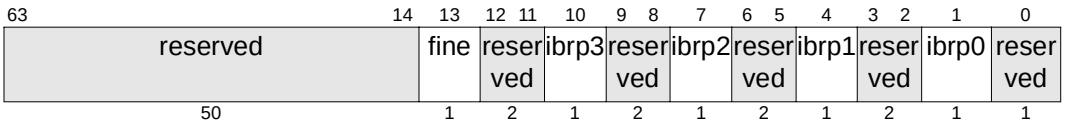


Table 10-12. Instruction Address Range Configuration Register Fields (PMC_{14})

Field	Bits	Description
ibrp0	1	1: No constraint 0: Non-prefetch PMU events (IA64_TAGGED_INST_RETIRED.00 included) will be constrained by $IBRP_0$
ibrp1	4	1: No constraint 0: Prefetch PMU events (IA64_TAGGED_INST_RETIRED.01 included) will be constrained by $IBRP_1$
ibrp2	7	1: No constraint 0: Non-prefetch PMU events (IA64_TAGGED_INST_RETIRED.10 included) ill be constrained by $IBRP_2$
ibrp3	10	1: No constraint 0: Non-prefetch PMU events (IA64_TAGGED_INST_RETIRED.11 included) will be constrained by $IBRP_3$
fine	13	Enable arbitrary range checking (non power of 2) 1: $IBRP_{0,2}$ and $IBRP_{1,3}$ are paired as lo/hi limit bits 0: Normal mode This bit provides this capability. If set to 1, ibrp0 (lower limit) and ibrp2 (upper limit) are paired together; So are ibrp1 (lower limit) and ibrp3(upper limit). Bits [63:12] of upper and lower limits need to be exactly the same but could have any value. Bits[11:0] of upper limit needs to be greater than bits[11:0] of lower limit. If an address falls in between the upper and lower limits then a match will be signaled for both of the ibr pairs used (ibrp0 and ibrp2 will signals matches at the same time). NOTE: The mask bits programmed in IBRs 1,3,5,7 for bits [11:0] have no effect in this mode.

$IBRP_0$ 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 PMC14.Fine=0, IBRmatch0 = match[IP(63:0), IBR0(63:0), IBR1(55:0)]
Else, IBRmatch0 = match[IP(63:12), IBR0(63:12), IBR1(55:12)] and [IP(11:0) >
IBR0(11:0)] and [IP(11:0) < IBR4(11:0)]
     IBRadrmatch0 = IBRmatch0
     ibrp0 match = (PMC8.ign or PMC14.ibrp0) or (IBRadmatch0 and match[PSR.cpl,
IBR1(59:56)])
```

The instruction range check tag (IBRRangeTag) considers the IBR address ranges only if PMC_8.ig_ad, PMC_{14}.ibrp0 and `PSR.is` are all zero and if none of the IBR x-bits or PSR.db are set.

In order to allow simultaneous use of some IBRs for Performance Monitoring and the others for debugging (the architected purpose of these registers), separate mechanisms are provided for enabling IBRs and the x-bit should be cleared to 0 for the IBRP which is going to be used for PMU.

10.3.5.1 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_{14} must contain the predetermined value 0xdb6. If software modifies any bits not listed in Table 10-12 processor behavior is not defined. It is illegal to have PMC_{13}[48:45]=0000 and PMC_8[0]=0 and ((PMC_{14}[2:1]=10 or 00) or (PMC_{14}[5:4]=10 or 00)); this produces inconsistencies in tagging I-side events in L1D and L2.

10.3.5.2 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 $IBR_{2,3}$.

10.3.6 Data Address Range Matching (PMC_{13})

For instructions that reference memory, the Itanium 2 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_{13}).

Figure 10-17 and Table 10-11 describe the fields of register $PMC_{13.}$ 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.

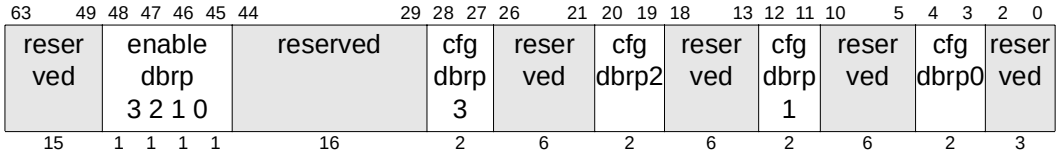
Table 10-13. Memory Pipeline Event Constraints Fields (PMC_{13})

Field	Bits	Description
cfg dbrp0	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 $IBRP_0$/PMC_8 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 $IBRP_0$/PMC_8 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, cfg_dbrp0 needs to be set according to the desired constraints but cfg_dbrp2 should be left as 11 (No constraints).
cfg dbrp1	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 $DBRP_0$.
cfg dbrp2	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 $DBRP_0$.
cfg dbrp3	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 DBRP0.
Enable dbrp0	45	0 - No constraints 1 - Constraints as set by cfg dbrp0.

Table 10-13. Memory Pipeline Event Constraints Fields (PMC_{13}) (Continued)

Field	Bits	Description
Enable dbrp1	46	0 - No constraints 1 - Constraints as set by cfg dbrp1
Enable dbrp2	47	0 - No constraints 1 - Constraints as set by cfg dbrp2
Enable dbrp3	48	0 - No constraints 1 - Constraints as set by cfg dbrp3

Figure 10-17. Memory Pipeline Event Constraints Configuration Register (PMC_{13})



$DBRP_x$ 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 with PSR.db to form a single DBR match:

```
DBRRangeMatch = ((DBRRangeMatch0 or DBRRangeMatch1 or DBRRangeMatch2 or
DBRRangeMatch3) and (not PSR.db))
```

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_{13} 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 and the r/w-bit should be cleared to 0 for the DBRP which is going to be used for the PMU.

Note: Register PMC_{13} must contain the predetermined value 0x2078fefefefe. If software modifies any bits not listed in Table 10-11 processor behavior is not defined. It is illegal to have PMC13[48:45]=0000 and PMC_8[0]=0 and ((PMC_{14}[2:1]=10 or 00) or (PMC_{14}[5:4]=10 or 00)); this produces inconsistencies in tagging I-side events in L1D and L3.

10.3.7 Event Address Registers ($PMC_{10,11}$/$PMD_{0,1,2,3,17}$)

This section defines the register layout for the Itanium 2 processor instruction and data event address registers (EARs). Sampling of six events is supported on the Itanium 2 processor: instruction cache and instruction TLB misses, data cache load misses and data TLB misses, ALAT misses, and front-end stalls. The EARs are configured through two PMC registers ($PMC_{10,11}$). EAR specific unit masks allow software to specify event collection parameters to hardware. Instruction and data addresses, operation latencies and other captured event parameters are provided in five PMD registers ($PMD_{0,1,2,3,17}$). The instruction and data cache EARs report the latency of captured cache events and allow latency thresholding to qualify event capture. Event address data registers ($PMD_{0,1,2,3,17}$) contain valid data only when event collection is frozen (PMC_0.fr is one). Reads of $PMD_{0,1,2,3,17}$ while event collection is enabled return undefined values.

10.3.7.1 Instruction EAR (PMC_{10}, $PMD_{0,1}$)

The instruction event address configuration register (PMC_{10}) can be programmed to monitor either L1 instruction cache or instruction TLB miss events. Figure 10-18 and Table 10-14 detail the register layout of PMC_{10}. Table describes the associated event address data registers $PMD_{0,1}$.

Figure 10-18. Instruction Event Address Configuration Register (PMC_{10})

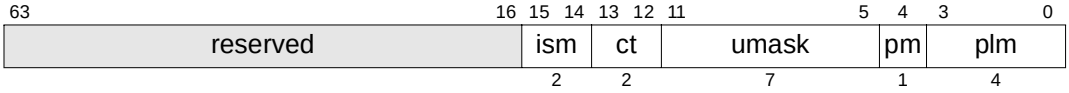
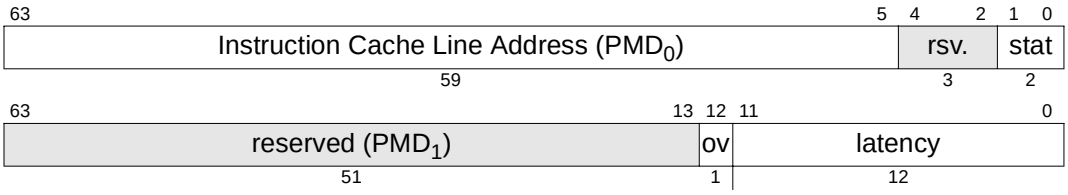


Table 10-14. Instruction Event Address Configuration Register Fields (PMC_{10})

Field	Bits	Description	
plm	3:0	See Table 10-5, “Performance Monitor PMC Register Control Fields (PMC4,5,6,7, 0,11,12)”	
pm	4	See Table 10-5, “Performance Monitor PMC Register Control Fields (PMC4,5,6,7, 0,11,12)”	
umask	11:5 12:5	Selects the event to be monitored If [13] = ‘1 then [12:5] are used for umask	
ct	13:12	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. $PMD_{0,1}$ register interpretation (see Table 10-16, “Instruction EAR (PMD0,1) in Cache Mode (PMC10.ct=’1x)”)
		if =01:	Nothing monitored
		if =00:	Monitor L1 instruction TLB misses $PMD_{0,1}$ register interpretation (see Table 10-16, “Instruction EAR (PMD0,1) in Cache Mode (PMC10.ct=’1x)”)
ism	15:14	See Table 10-5, “Performance Monitor PMC Register Control Fields (PMC4,5,6,7, 0,11,12)”	
ignored	31:16 47:32	Will each return value of bits[15:0] when read	

Figure 10-19. Instruction Event Address Register Format ($PMD_{0,1}$)



When the cache_tlb-bit (PMC_{10}.ct) is set to zero, instruction cache misses are monitored. When it is set to one, instruction TLB misses are monitored. The interpretation of the umask field and performance monitor data registers $PMD_{0,1}$ depends on the setting of this bit and is described in Section 10.3.7.2, “Instruction EAR Cache Mode (PMC10.ct=’1x)” for instruction cache monitoring and in Section 10.3.7.3, “Instruction EAR TLB Mode (PMC10.ct=00)” for instruction TLB monitoring.

10.3.7.2 Instruction EAR Cache Mode (PMC_{10}.ct='1x)

When PMC_{10}.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 a four bit umask field in the configuration register PMC_{10}. Possible threshold values are defined in Table 10-15.

Table 10-15. Instruction EAR (PMC_{10}) umask Field in Cache Mode (PMC_{10}.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	--------	>=512
11111110	>=8	11000000	>=1024
11111100	>=16	--------	>=2048
11111000	>=32	10000000	>=4096
--------	>=64	other	undefined
11110000	>=128	00000000	RAB hit (All L1 misses which hit in RAB)

As defined in Table 10-16, the address of the instruction cache line missed the L1 instruction cache is provided in PMD_0. If no qualified event was captured, the valid bit in PMD_0 is zero. The latency of the captured instruction cache miss in CPU clock cycles is provided in the latency field of PMD_1. In cache mode, the TLB miss bit of PMD_0 is undefined.

Table 10-16. Instruction EAR ($PMD_{0,1}$) in Cache Mode (PMC_{10}.ct='1x)

Register	Field	Bits	Description
PMD0	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_1	latency	11:0	Latency in CPU clocks
	overflow	12	If 1, latency counter has overflowed one or more times before data was returned

10.3.7.3 Instruction EAR TLB Mode (PMC_{10}.ct=00)

When PMC_{10}.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 10-17 summarizes the instruction TLB umask settings. All combinations of the mask bits are supported.

Table 10-17. Instruction EAR (PMC_{10}) umask Field in TLB Mode (PMC_{10}.ct=00)

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 10-17. Instruction EAR (PMC_{10}) umask Field in TLB Mode (PMC_{10}.ct=00) (Continued)

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 10-18 the address of the instruction cache line fetch that missed the L1 ITLB is provided in PMD_0. The stat bit [1] indicates whether the captured TLB miss hit in the VHPT or required servicing by software. If no qualified event was captured, the valid bit in PMD_0 reads zero. In TLB mode, the latency field of PMD_1 is undefined.

Table 10-18. Instruction EAR ($PMD_{0,1}$) in TLB Mode (PMC_{10}.ct='00)

Register	Field	Bits	Description
PMD_0	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_1	latency	11:2	Undefined in TLB mode

10.3.8 Data EAR (PMC_{11}, $PMD_{2,3,17}$)

The data event address configuration register (PMC_{11}) can be programmed to monitor either L1 data cache load misses, FP loads, L1 data TLB misses, or ALAT misses. Figure 10-20 and Table 10-19 detail the register layout of PMC_{11}. Figure 10-21 describes the associated event address data registers $PMD_{2,3,17}$. The mode bits in configuration register PMC_{11} select data cache, data TLB, or ALAT monitoring. The interpretation of the umask field and registers $PMD_{2,3,17}$ depends on the setting of the mode bits and is described in Section 10.3.8.1, "Data Cache Load Miss Monitoring (PMC11.mode=00)" for data cache load miss monitoring, Section 10.3.8.2, "Data TLB Miss Monitoring (PMC11.mode='01)" for data TLB monitoring, and Section 10.3.8.3, "ALAT Miss Monitoring (PMC11.mode='1x)" for ALAT monitoring.

Figure 10-20. Data Event Address Configuration Register (PMC_{11})

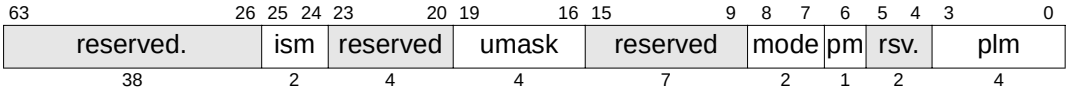


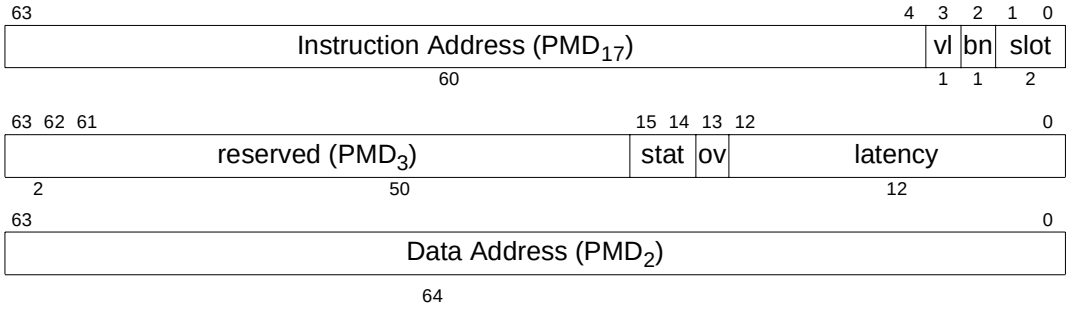
Table 10-19. Data Event Address Configuration Register Fields (PMC_{11})

Field	Bits	Description
plm	3:0	See Table 10-5 "Performance Monitor PMC Register Control Fields (PMC4,5,6,7, 0,11,12)."
pm	6	See Table 10-5 "Performance Monitor PMC Register Control Fields (PMC4,5,6,7, 0,11,12)."

Table 10-19. Data Event Address Configuration Register Fields (PMC_{11}) (Continued)

Field	Bits	Description
mode	8:7	Data EAR mode selector: ‘00: L1 data cache load misses and FP loads ‘01: L1 data TLB misses ‘1x: ALAT misses
umask	19:16	Data EAR unit mask mode 00: data cache unit mask (definition see Table 10-20, “Data EAR (PMC11) Umask Fields in Data Cache Mode (PMC11.mode=00)”) mode 01: data TLB unit mask (definition see Table 10-22, “Data EAR (PMC11) Umask Field in TLB Mode (PMC10.ct=01)”)
ism	25:24	See Table 10-5 “Performance Monitor PMC Register Control Fields (PMC4,5,6,7, 0,11,12).”

Figure 10-21. Data Event Address Register Format ($PMD_{2,3,17}$)



10.3.8.1 Data Cache Load Miss Monitoring (PMC_{11}.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 10-20.

As defined in Table 10-22, 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 10-20. Data EAR (PMC_{11}) Umask Fields in Data Cache Mode (PMC_{11}.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 10-21. $PMD_{2,3,17}$ Fields in Data Cache Load Miss Mode (PMC_{11}.mode=00)

Register	Fields	Bit Range	Description
PMD_2	Data Address	63:0	64-bit virtual address of data item that caused miss
PMD_3	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 PMD2,17 and rest of PMD3 01: Valid information in PMD2,3 and may be in PMD17 NOTE: These bits should be cleared before the EAR is reused.
PMD_{17}	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 Itanium 2 processor data cache miss event address register can only track a single load at a time, not all data cache load misses may be captured. When the processor hardware captures the address of a load (called the monitored load), it ignores all other overlapped concurrent loads until it is determined whether the monitored load turns out to be an L1 data cache miss or not. If the monitored load turns out to be a cache miss, its parameters are latched into $PMD_{2,3,17}$. The processor randomizes the choice of which load instructions are tracked to prevent the same data cache load miss from always being captured (in a regular sequence of overlapped data cache load misses). While this mechanism will not always capture all data cache load misses in a particular sequence of overlapped loads, its accuracy is sufficient to be used by statistical sampling or code instrumentation.

10.3.8.2 Data TLB Miss Monitoring (PMC_{11}.mode='01)

If the Data EAR is configured to monitor data TLB misses, the umask defined in Table 10-23 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 10-23 the instruction and data addresses of captured DTLB misses are presented to software in $PMD_{2,17}$. If no qualified event was captured, the valid bit in PMD_{17} reads zero. When programmed for data TLB monitoring, the contents of the latency field of PMD_3 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_{12} must be 0 in this mode; else the wrong IP for misses coming right after a mispredicted branch.

Table 10-22. Data EAR (PMC_{11}) Umask Field in TLB Mode (PMC_{10}.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 10-23. $PMD_{2,3,17}$ Fields in TLB Miss Mode (PMC_{11}.mode='01)

Register	Field	Bit Range	Description
PMD_2	Data Address	63:0	64-bit virtual address of data item that caused miss
PMD_3	latency	12:0	Undefined in TLB Miss mode
	ov	13	Undefined in TLB Miss mode
	stat	15:14	Status 00: invalid information in PMD2,17 and rest of PMD3 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.
PMD_{17}	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.

10.3.8.3 ALAT Miss Monitoring (PMC_{11}.mode='1x)

As defined in Table 10-24, the address of the instruction (failing `chk.a` and `ld.c`) causing an ALAT miss is presented to software in PMD_{17}. If no qualified event was captured, the valid bit in PMD_{17} reads zero. When programmed for ALAT monitoring, the latency field of PMD_3 and the contents of PMD_2 are undefined.

Note: PMC_{12} must be 0 in this mode; else the wrong IP for misses coming right after a mispredicted branch.

Table 10-24. $PMD_{2,3,17}$ Fields in ALAT Miss Mode (PMC_{11}.mode='1x)

Register	Field	Bit Range	Description
PMD_2	Data Address	63:0	Undefined in ALAT Miss Mode
PMD_3	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 $PMD_{2,17}$ and rest of PMD_3 01: Valid information in $PMD_{2,3}$ and may be in PMD_{17} NOTE: These bits should be cleared before the EAR is reused.
PMD_{17}	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.

10.3.9 Branch Trace Buffer

The branch trace buffer provides information about the outcome of the most recent Itanium branch instructions and their predictions and outcomes. The Itanium 2 branch trace buffer configuration register (PMC_{12}) defines the conditions under which branch instructions are captured, and allows the trace buffer to capture specific subsets of branch events. The branch trace buffer operates only during Itanium-based code execution, i.e. when `PSR.is` is zero. When running IA-32 ISA, the branch trace buffer is not updated.

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

10.3.9.1 Branch Trace Buffer Collection Conditions

The branch trace buffer configuration register (PMC_{12}) defines the conditions under which branch instructions are captured. These conditions are given in Figure 10-22 and Table 10-25, which refer to conditions associated with the branch prediction. These conditions are:

- Whether the target of the branch should be captured or additional information about the prediction 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

Figure 10-22. Branch Trace Buffer Configuration Register (PMC_{12})

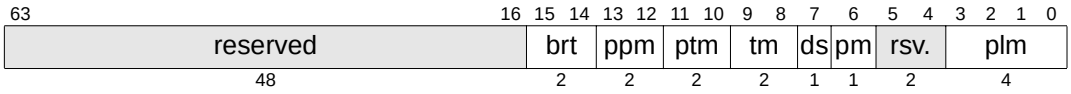


Table 10-25. Branch Trace Buffer Configuration Register Fields (PMC_{12})

Field	Bits	Description
plm	3:0	See Table 10-5, “Performance Monitor PMC Register Control Fields (PMC4,5,6,7, 0,11,12)”
pm	6	See Table 10-5, “Performance Monitor PMC Register Control Fields (PMC4,5,6,7, 0,11,12)”
ds	7	Data selector: 1: capture info about branch predictions 0: capture branch target
tm	9:8	Taken Mask: 11: all Itanium® branches 10: Taken Itanium branches only 01: Not Taken Itanium 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

To summarize, an Itanium 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 branches, the Itanium 2 branch trace buffer configuration settings in PMC_{12} 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 10.3.5, "Instruction Address Range Matching") and Opcode Matching (Section 10.3.4, "Opcode Match Check (PMC8,9,15)") may also be used to constrain what is captured in the Branch Trace Buffer.

10.3.9.2 Branch Trace Buffer Reading

Figure 10-23. Branch Trace Buffer Register Format (PMD_{8-15}, where PMC_{12}.ds == 0)

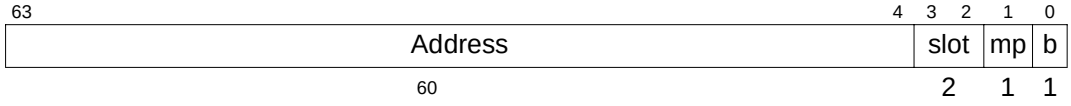


Figure 10-24. Branch Trace Buffer Register Format (PMD_{8-15}, where PMC_{12}.ds == 1)

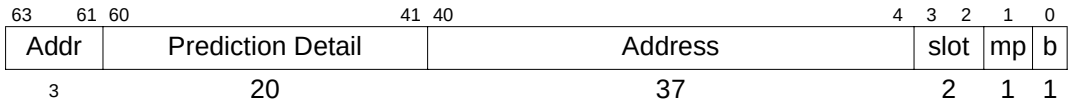


Table 10-26. Branch Trace Buffer Register Fields (PMD_{8-15})

Field	Bit Range	Description
b	0	Branch Bit 1: contents of register is a branch instruction 0: contents of register is a branch target or contains branch prediction detail
mp	1	Mispredict Bit if b=1 and mp=1: mispredicted branch (due to target or predicate misprediction) if b=1 and mp=0: correctly predicted branch if b=0 and mp=1: valid target address if b=0 and mp=0: invalid branch trace buffer register
slot	3:2	if b=0: undefined if b=1: Slot index of first taken branch instruction in bundle 00: Itanium Slot 0 branch/target 01: Itanium Slot 1 branch/target 10: Itanium Slot 2 branch/target 11: this was a not taken branch
Address	63:4	if b=1: 60-bit bundle address of Itanium branch instruction if ds=0 and b=0: 60-bit target bundle address of Itanium branch instruction if ds=1 and b=0: Upper 3 bits and lower 37 bits of the bundle address of Itanium® branch instruction and the lower 20 bits of the L1 IBR associated with the captured branch

The eight branch trace buffer registers PMD_{8-15} provide information about the outcome of a captured branch sequence. The branch trace buffer registers (PMD_{8-15}) contain valid data only when event collection is frozen (PMC_0.fr is one). While event collection is enabled, reads of PMD_{8-15} return undefined values. The registers follow the layout defined in Figure 10-23, Figure 10-24, and Table 10-26 contain the address of either a captured branch instruction (b-bit=1) or a branch target (b-bit=0) or branch prediction details. For branch instructions, the mp-bit indicates a branch misprediction. A branch trace register with a zero b-bit and a zero mp-bit indicates an invalid branch trace buffer entry. The slot field captures the slot number of the first taken Itanium branch instruction in the captured instruction bundle. A slot number of 3 indicates a not-taken branch. The target address bundle of a branch to IA-32 (`br.ia`) is recorded. An IA-32 JMPE branch instruction and its Itanium target are not recorded.

In every cycle in which a qualified Itanium 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_{12}.ds=0) or details of the branch prediction (PCM_{12}.ds=1) are written to the next buffer location. As a result, the branch trace buffer may contain a mixed sequence of the branches and targets.

In order to be able to record more information in the trace buffer, there are two cases which will not have the branch target/prediction history recorded:

- Taken IP-relative branches with PMC12.ds == 0
- Not-Taken branches with PMC12.ds == 0

The Itanium 2 branch trace buffer is a circular buffer containing the last four to eight qualified Itanium branches. The Branch Trace Buffer Index Register (PMD_{16}) defined in Figure 10-25 and Table 10-27 identify the most recently recorded branch or target. In every cycle in which a qualified branch or target is recorded, the branch buffer index (bbi) is post-incremented. After 8 entries have been recorded, the branch index wraps around, and the next qualified branch will

1. In some cases, the Itanium® 2 processor branch trace buffer will capture the source (but not the target) address of an excepting branch instruction. This occurs on trapping branch instructions as well as faulting `br.ia`, `break.b` and multi-way branches.

overwrite the first trace buffer entry. The wrap condition itself is recorded in the full bit of PMD_{16}. The bbi field of PMD_{16} defines the next branch buffer index that is about to be written.The following formula computes the last written branch trace buffer PMD index from the contents of PMD_{16}:

$$\text{last-written-PMD-index} = 8+ ([\ (8*PMD_{16}.\text{full}) + (PMC_{16}.\text{bbi} - 1)] \ \% \ 8)$$

If both the full bit and the bbi field of PMD_{16} are zero, no qualified branch has been captured by the branch trace buffer. The full bit gets set the every time the branch trace buffer wraps from PMD_{15} to PMD_8. Once set, the full bit remains set until explicitly cleared by software, i.e. it is a sticky bit. Software can reset the bbi index and the full bit by writing to PMD_{16}.

Figure 10-25. Branch Trace Buffer Index Register Format (PMD_{16})

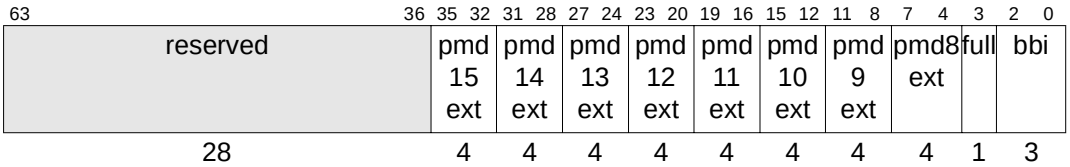


Table 10-27. Branch Trace Buffer Index Register Fields (PMD_{16})

Field	Bit Range	Description
bbi	2:0	Branch Buffer Index [Range 0..7 - Index 0 indicates PMD_8] Pointer to the next branch trace buffer entry to be written if full=1: points to the oldest recorded branch/target if full=0: points to the next location to be written
full	3	Full Bit (sticky) if full=1: branch trace buffer has wrapped if full=0: branch trace buffer has not wrapped
pmd8 ext	7:4	bit[7:6] Not used bit[5] (bruflush): If PMD8.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 bit[4] (b1): if b = 1 1 = branch was from bundle 1, add 0x1 to PMD8.bits[63:4] 0 = branch was from bundle 0
pmd9 ext	11:8	Same as above for PMD9
pmd10 ext	15:12	Same as above for PMD10
pmd11 ext	19:16	Same as above for PMD11
pmd12 ext	23:20	Same as above for PMD12
pmd13 ext	27:24	Same as above for PMD13
pmd14 ext	31:28	Same as above for PMD14
pmd15 ext	35:32	Same as above for PMD15

10.3.10 Interrupts

As mentioned in Table 10-6, each one of registers $PMD_{4,5,6,7}$ 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.

10.3.10.1 External Events

As mentioned in theTable 10-6, each PMD will cause an external event on the BPM# pin if the following conditions are all true: Currently the signal will reflect the value of the overflow bit of the PMD [47]. Meaning once it made a 0 to 1 transition, it will make a 1 to 0 transition either when the PMD was re-written with bit W=0 or when PMD overflows one more time.

- PMC_i.ev=1 (i.e. external event is enabled for PMD_i) and PMD_i overflows (read bit 47 of PMD_i=1). This pin will stay high as long as these conditions are true.
- BPM[5:0] are bidirectional processor pins allocated for debug and performance monitor. The exact method of enabling these pins is not known at this time. But there will be a way to determine their direction (in versus out) and which information will show up on them (output) or how the information will be used (input).

10.3.11 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:

- $PMC_{8,9}$ = 0xffffffffffffffff, (match all opcodes)
- PMC_{13} = 0x2078fefefefe, (no memory pipeline event constraints)
- PMC_{14} = 0xdb6, (no instruction address range constraints)
- PMC_{15} = 0xffffff0, (no opcode match constraints)

All other performance monitoring related state is undefined.

Table 10-28. Information Returned by PAL_PERF_MON_INFO for the Itanium® 2 Processor

PAL_PERF_MON_INFO Return Value	Description	Itanium® 2 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 in 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 Itanium 2 processor specific values are summarized in Table 10-28.

Low Power State: On the Itanium 2 processors, PAL_HALT_LIGHT selectively freezes specific performance monitoring events in order to preserve them prior to powering down the processor. Below is a list of performance monitors that will continue to be monitored while all others are frozen:

- BUS_ALL.IO
- BUS_ALL.ANY
- BUS_DATA_CYCLE
- BUS_IO.IO
- BUS_IO.ANY
- BUS_LOCK.ANY
- BUS_MEMORY.EQ_128BYTE.IO
- BUS_MEMORY.EQ_128BYTE.ANY
- BUS_MEMORY.LT_128BYTE.IO

- BUS_MEMORY.LT_128BYTE.ANY
- BUS_MEMORY.ALL_128BYTE.IO
- BUS_MEMORY.ALL_128BYTE.ANY
- BUS_MEM_READ.BIL.IO
- BUS_MEM_READ.BIL.ANY
- BUS_MEM_READ.BRL.IO
- BUS_MEM_READ.BRL.ANY
- BUS_MEM_READ.BRIL.IO
- BUS_MEM_READ.BRIL.ANY
- BUS_MEM_READ.ALL.IO
- BUS_MEM_READ.ALL.ANY
- BUS_RD_DATA.IO
- BUS_RD_DATA.ANY
- BUS_RD_IO.IO
- BUS_RD_IO.ANY
- BUS_RD_PRTL.IO
- BUS_RD_PRTL.ANY
- BUS_SNOOPS.IO
- BUS_SNOOPS.ANY
- BUS_SNOOPS_HITM.ANY
- BUS_SNOOP_STALL_CYCLES.ANY
- BUS_WR_WB.EQ_128BYTE.IO
- BUS_WR_WB.EQ_128BYTE.ANY
- BUS_WR_WB.CCASTOUT.ANY
- BUS_WR_WB.ALL.IO
- BUS_WR_WB.ALL.ANY
- L1I_PURGE
- MEM_READ_CURRENT.IO
- MEM_READ_CURRENT.ANY

Performance Monitor Events 11

11.1 Introduction

This chapter describes the architectural and microarchitectural events measurable on the Itanium 2 processor through the performance monitoring mechanisms described earlier in Chapter 10. 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 11, "Categorization of Events."

The Itanium 2 processor is capable of monitoring numerous events. The majority of events can be selected as input to any of the PMD_{4-7} 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 11.8.2 and Section 11.8.3 for events that have more specific requirements.

11.2 Categorization of Events

Performance related events are grouped into the following categories:

- Basic Events: Clock cycles, retired instructions (Section 11.3)
- Instruction Dispersal Events: Instruction decode and issue (Section 11.4)
- Instruction Execution Events: Instruction execution, data and control speculation, and memory operations (Section 11.5)
- Stall Events: Stall and execution cycle breakdowns (Section 11.6)
- Branch Events: Branch prediction (Section 11.7)
- Memory Hierarchy: Instruction and data caches (Section 11.8)
- System Events: Operating system monitors (Section 11.9)
- TLB Events: Instruction and data TLBs (Section 11.10)
- System Bus Events: (Section 11.11)
- RSE Events: Register Stack Engine (Section 11.12)

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 11.14.

Directly measurable events often use the PMC.umask field (See Section 10.3.2, "Performance Counter Registers") 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.
- Description - Brief description of the event.

11.3 Basic Events

Table 11-1 summarizes two basic execution monitors. The CPU_CYCLES event can be used to break out separate or combined Itanium architecture/IA-32 cycle counts by constraining the PMC/PMD based on the currently executing instruction set. The Itanium 2 retired instruction count, IA64_INST_RETIRED, includes predicated true instructions and nop instructions, but excludes RSE operations.

Table 11-1. Performance Monitors for Basic Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
CPU_CYCLES	0x12	N	N	N	1	CPU Cycles
IA64_INST_RETIRED	0x08	Y	N	Y	6	Retired Itanium® Instructions
IA32_INST_RETIRED	0x59	N	N	N	2	IA-32 Instructions Retired
IA32_ISA_TRANSITIONS	0x07	N	N	N	1	Itanium to/from IA-32 ISA Transitions

Table 11-2. Derived Monitors for Basic Events

Symbol Name	Description	Equation
IA64_IPC	Average Number of Itanium® Instructions Per Cycle During Itanium-based Code Sequences	IA64_INST_RETIRED / CPU_CYCLES
IA32_IPC	Average Number of IA-32 Instructions Per Cycle During IA-32 Code Sequences	IA32_INST_RETIRED / CPU_CYCLES
AVG_CPT	Average Number of Cycles per ISA Transition	CPU_CYCLES / (ISA_TRANSITIONS * 2)
AVG_IA32_IPT	Average Number of IA-32 Instructions per ISA Transition	IA32_INST_RETIRED / (IA32_ISA_TRANSITIONS / 2)
AVG_IA64_IPT	Average Number of Itanium Instructions per ISA Transition	IA64_INST_RETIRED / (IA32_ISA_TRANSITIONS / 2)

11.4 Instruction Dispersal Events

Instruction cache lines are delivered to the execution core and dispersed to the Itanium 2 processor functional units. The Itanium 2 processor can issue, or disperse, 6 instructions per clock cycle. In other words, the Itanium 2 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_CYCLES-DISP_STALLED) = INST_DISPERSED + SYLL_NOT_DISPERSED.ALL - SYLL_OVERCOUNT.ALL

Table 11-3. 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

11.5 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 11-4 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 Itanium 2 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 11-4. 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	8	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
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 11-5. 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

11.6 Stall Events

Itanium 2 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 Itanium 2 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, the back-end pipeline has either a bubble or it retires 1 or more instructions, CPU_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.

Each of the stall events (summarized in Table 11-6) take a umask to choose among several available sub-events. Please refer to the detailed event descriptions in Section 11.14 for a list of available sub-events and their individual descriptions.

Table 11-6. 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

11.7 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 Branch Trace Buffer (also known as Branch EARs). For detailed information on the Branch EARs please refer to Section 10.3.9, “Branch Trace Buffer”.

Table 11-7. 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	1	Number of encoded branches retired

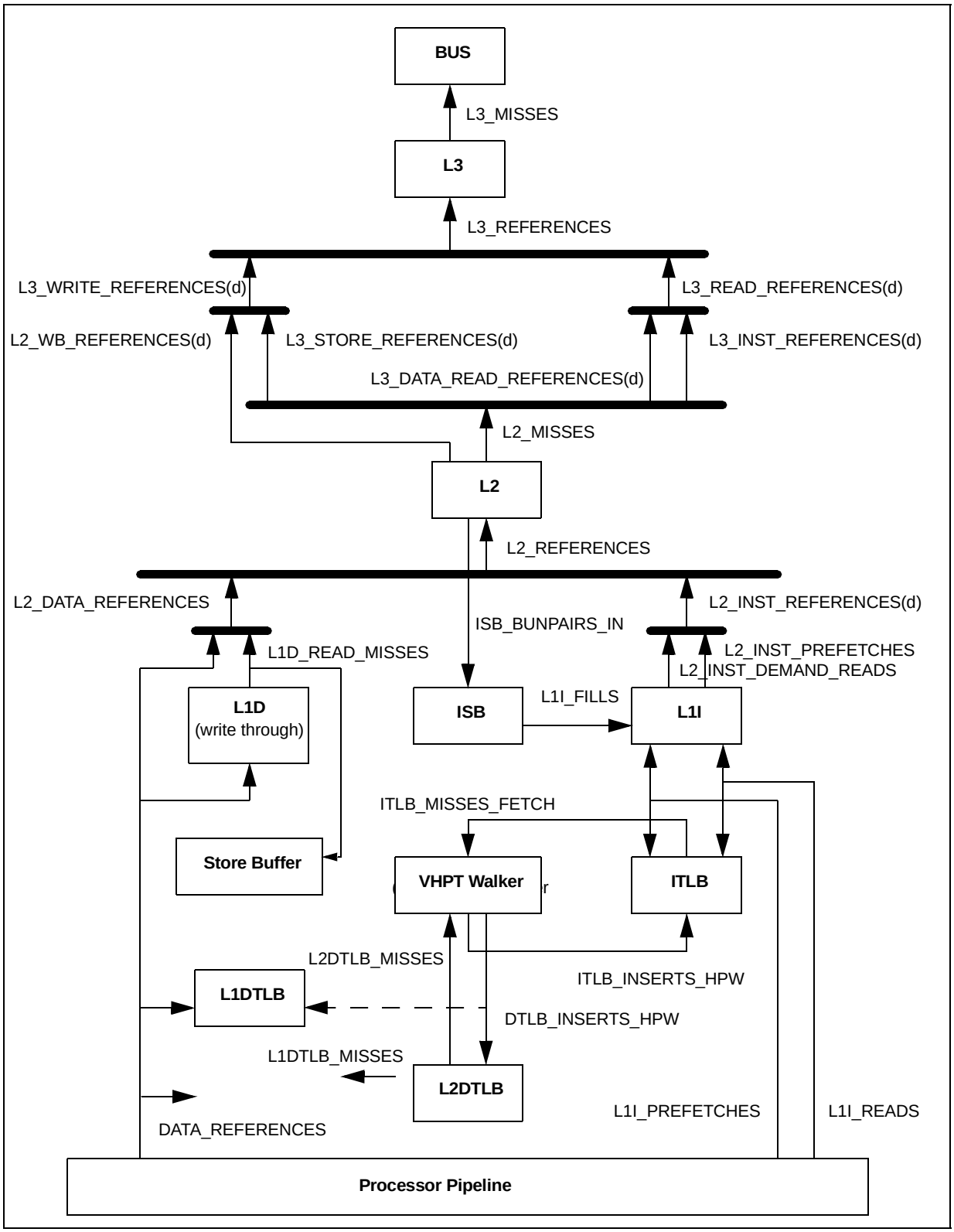
11.8 Memory Hierarchy

This section summarizes events related to the Itanium 2 processor’s memory hierarchy. The memory hierarchy events are grouped as follows:

- L1 Instruction Cache and Prefetch Events (Section 11.8.1)
- L1 Data Cache Events (Section 11.8.2)
- L2 Unified Cache Events (Section 11.8.3)
- L3 Cache Events (Section 11.8.4)

An overview of the Itanium 2 processor’s three level memory hierarchy and its event monitors is shown in Figure 11-1. The instruction and the data stream work through separate L1 caches. The L1 data cache is a write-through cache. A unified L2 cache serves both the L1 instruction and data caches, and is backed by a large unified L3 cache. Events for individual levels of the cache hierarchy are described in the following three sections.

Figure 11-1. Event Monitors in the Itanium® 2 Processor Memory Hierarchy



11.8.1 L1 Instruction Cache and Prefetch Events

Table 11-8 describes and summarizes the events that the Itanium 2 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 L2 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 Section 10.3.5, “Instruction Address Range Matching” for details).

L1I_EAR_EVENTS counts the number of events captured by the Itanium 2 processor’s instruction EARs. Please refer to Section 10.3.7, “Event Address Registers (PMC10,11/PMD0,1,2,3,17)” for more detailed information about the instruction EARs.

Table 11-8. Performance Monitors for L1 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 L2 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
L2_INST_DEMAND_READS	0x42	Y	N	N	1	L1 Instruction Cache and ISB Misses
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
L2_INST_PREFETCHES	0x45	Y	N	N	1	L2 Instruction Prefetch Requests

Table 11-9. Derived Monitors for L1 Instruction Cache and Prefetch Events

Symbol Name	Description	Equation
ISB_LINES_IN	Number of cache lines written from L2 (and beyond) into the front-end	ISB_BUNPAIRS_IN/4
L1I_DEMAND_MISS_RATIO	L1I Demand Miss Ratio	L2_INST_DEMAND_READS / L1I_READS
L1I_PREFETCH_MISS_RATIO	L1I Prefetch Miss Ratio	L2_INST_PREFETCHES / L1I_PREFETCHES
L1I_REFERENCES	Number of L1 Instruction Cache reads and fills	L1I_READS + L1I_PREFETCHES

11.8.2 L1 Data Cache Events

Table 11-10 lists the Itanium 2 processor's L1 data cache monitors. As shown in Figure 11-1, 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 10.3.8, "Data EAR (PMC11, PMD2,3,17)" for more detailed information about the data EARs.

L1D cache events have been divided into five sets. 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 the following sections.

Table 11-10. 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

11.8.2.1 L1D Cache Events (Set 0)

Table 11-11. 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

11.8.2.2 L1D Cache Events (Set 1)

Table 11-12. 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

11.8.2.3 L1D Cache Events (Set 2)

Table 11-13. 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

11.8.2.4 L1D Cache Events (Set 3)

Table 11-14. 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

11.8.2.5 L1D Cache Events (Set 4)

Table 11-15. 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

11.8.3 L2 Unified Cache Events

Table 11-16 summarizes the events available to monitor the Itanium 2 processor L2 cache.

L2 cache events have been divided into 6 sets. Only events within a set (or non-L2 events) can be measured at the same time. Each set is selected by the event code programmed into PMC4 (i.e. if you want to measure any of the events in this set, one of them needs to be measured by PMD4). Within a set, certain events can only be measured by PMD4. There may also be some limitations on umasks in which the prime event (the L2 event using PMD4) will dictate the umask for certain companion L2 events. These will be noted by set. Monitors belonging to each set are explicitly presented in the following sections.

Table 11-16. Performance Monitors for L2 Unified Cache Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2_BAD_LINES_SELECTED	0xb9	Y	Y	Y	4	Valid line replaced when invalid line is available
L2_BYPASS	0xb8	Y	Y	Y	1	Count bypass
L2_DATA_REFERENCES	0xb2	Y	Y	Y	4	Data RD/WR access to L2
L2_FILLB_FULL	0xbf	N	N	N	1	L2D Fill buffer is full
L2_FORCE_RECIRC	0xb4	Y	Y	Y	4	Forced recirculates
L2_GOT_RECIRC_IFETCH	0xba	Y	Y	Y	1	Instruction fetch recirculates received by L2D
L2_GOT_RECIRC_OZQ_ACC	0xb6	Y	Y	Y	1	Counts number of OZQ accesses recirculated back to L1D
L2_IFET_CANCELS	0xa1,0xa5, 0xa9, 0xad	Y	Y	Y	1	Instruction fetch cancels by the L2
L2_ISSUED_RECIRC_IFETCH	0xb9	Y	Y	Y	1	Instruction fetch recirculates issued by L2D
L2_ISSUED_RECIRC_OZQ_ACC	0xb5	Y	Y	Y	1	Count the number of times a recirculate issue was attempted and not preempted
L2_L3ACCESS_CANCEL	0xb0	Y	Y	Y	1	Canceled L3 accesses
L2_MISSES	0xcb	Y	Y	Y	1	L2 Misses
L2_OPS_ISSUED	0xb8	Y	Y	Y	4	Different operations issued by L2D
L2_OZDB_FULL	0xbd	N	N	N	1	L2D OZ data buffer is full

Table 11-16. Performance Monitors for L2 Unified Cache Events (Continued)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2_OZQ_ACQUIRE	0xa2,0xa6,0xaa,0xae	N	N	N	1	Clocks with acquire ordering attribute existed in L2 OZQ
L2_OZQ_CANCELS0	0xa0	Y	Y	Y	4	L2 OZQ cancels
L2_OZQ_CANCELS1	0xac	Y	Y	Y	4	L2 OZQ cancels
L2_OZQ_CANCELS2	0xa8	Y	Y	Y	4	L2 OZQ cancels
L2_OZQ_FULL	0xbc	N	N	N	1	L2D OZQ is full
L2_OZQ_RELEASE	0xa3,0xa7,0xab,0xaf	N	N	N	1	Clocks with release ordering attribute existed in L2 OZQ
L2_REFERENCES	0xb1	Y	Y	Y	4	Requests made from L2
L2_STORE_HIT_SHARED	0xba	Y	Y	Y	2	Store hit a shared line
L2_SYNTH_PROBE	0xb7	Y	Y	Y	1	Synthesize Probe
L2_VICTIMB_FULL	0xbe	N	N	N	1	L2D victim buffer is full

Table 11-17. Derived Monitors for L2 Unified Cache Events

Symbol Name	Description	Equation
L2_DATA_RATIO	Ratio of Data requests made to L2	L2_DATA_REFERENCES.L2_ALL / L2_REFERENCES
L2_DATA_READS	L2 Data Read Requests	L2_DATA_REFERENCES.L2_DATA_READS
L2_DATA_WRITES	L2 Data Write Requests	L2_DATA_REFERENCES.L2_DATA_WRITES
L2_INST_REFERENCES	Instruction requests made to L2	L2_INST_DEMAND_READS + L2_INST_PREFETCHES
L2_INST_FETCHES	Requests made to L2 due to demand instruction fetches	L2_INST_DEMAND_READS + L2_INST_PREFETCHES
L2_MISS_RATIO	Percentage of L2 Misses	L2_MISSES/L2_REFERENCES
L2_RECIRC_ATTEMPTS	Number of times the L2 issue logic attempted to issue a recirculate.	L2_ISSUED_RECIRC_OZQ_ACC + L2_OZQ_CANCELS2.DIDNT_RECIRC

A metric of interest is L2_MISS_RATIO; note that semaphores might cause this metric to be larger than 100% due to the fact that a semaphore will be counted once in L2_REFERENCES but may cause more than one L2_MISSES due to the cache line being snooped out and re-requested from the bus. This can be repeated many times until forward progress continues. Some level of error should be expected in this metric because L2_MISSES and L2_REFERENCES are lined up in time only for 5 cycle bypasses. (One can get around this problem by using the `mf.a` instruction followed by a `sync.i` followed by a `srlz.i` instruction before reading the counters).

11.8.3.1 L2 Cache Events (Set 0)

Either one of the L2_OZQ_CANCELS* events or L2_IFET_CANCELS must be measured by PMD4. These events use the same umask. Only 1 of the 3 L2_OZQ_CANCELS* events can be measured at any one time.

Table 11-18. Performance Monitors for L2 Cache Set 0

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2_IFET_CANCELS	0xa1,0xa5,0xa9,0xad	Y	Y	Y	1	Instruction fetch cancels by the L2.
L2_OZQ_ACQUIRE	0xa2,0xa6,0xaa,0xae	N	N	N	1	Clocks with acquire ordering attribute existed in L2 OZQ
L2_OZQ_CANCELS0	0xa0	Y	Y	Y	4	L2 OZQ cancels
L2_OZQ_CANCELS1	0xac	Y	Y	Y	4	L2 OZQ cancels
L2_OZQ_CANCELS2	0xa8	Y	Y	Y	4	L2 OZQ cancels
L2_OZQ_RELEASE	0xa3,0xa7,0xab,0xaf	N	N	N	1	Clocks with release ordering attribute existed in L2 OZQ

11.8.3.2 L2 Cache Events (Set 1)

L2_L3ACCESS_CANCEL must be measured by PMD4.

Table 11-19. Performance Monitors for L2 Cache Set 1

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2_DATA_REFERENCES	0xb2	Y	Y	Y	4	Data read/write access to L2
L2_L3ACCESS_CANCEL	0xb0	Y	Y	Y	1	Canceled L3 accesses
L2_REFERENCES	0xb1	Y	Y	Y	4	Requests made from L2

11.8.3.3 L2 Cache Events (Set 2)

L2_FORCE_RECIRC must be measured by PMD4.

Table 11-20. Performance Monitors for L2 Cache Set 2

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2_FORCE_RECIRC	0xb4	Y	Y	Y	4	Forced recirculates
L2_ISSUED_RECIRC_OZQ_ACC	0xb5	Y	Y	Y	1	Count number of times a recirculate issue was attempted and not preempted
L2_GOT_RECIRC_OZQ_ACC	0xb6	Y	Y	Y	1	Counts number of OZQ accesses recirculated back to L1D
L2_SYNTH_PROBE	0xb7	Y	Y	Y	1	Synthesized probe

11.8.3.4 L2 Cache Events (Set 3)

L2_BAD_LINES_SELECTED, L2_BYPASS, and L2_STORE_HIT_SHARED share the same umask.

Table 11-21. Performance Monitors for L2 Cache Set 3

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2_BAD_LINES_SELECTED	0xb9	Y	Y	Y	4	Valid line replaced when invalid line is available
L2_BYPASS	0xb8	Y	Y	Y	1	Count bypass
L2_STORE_HIT_SHARED	0xba	Y	Y	Y	2	Store hit a shared line

11.8.3.5 L2 Cache Events (Set 4)

Either one of L2_OPS_ISSUED, L2_ISSUED_RECIRC_IFETCH, or L2_GOT_RECIRC_IFETCH must be measured by PMD4. These three events share the same umask.

Table 11-22. Performance Monitors for L2 Cache Set 4

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2_GOT_RECIRC_IFETCH	0xba	Y	Y	Y	1	Instruction fetch recirculates received by L2D
L2_ISSUED_RECIRC_IFETCH	0xb9	Y	Y	Y	1	Instruction fetch recirculates issued by L2D
L2_OPS_ISSUED	0xb8	Y	Y	Y	4	Different operations issued by L2D

11.8.3.6 L2 Cache Events (Set 5)

Either one of L2_OZQ_FULL, L2_OZDB_FULL, L2_VICTIMB_FULL, or L2_FILLB_FULL must be measured by PMD4. These four events share the same umask.

Table 11-23. Performance Monitors for L2 Cache Set 5

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
L2_OZQ_FULL	0xbc	N	N	N	1	L2D OZQ is full
L2_OZDB_FULL	0xbd	N	N	N	1	L2D OZ data buffer is full
L2_VICTIMB_FULL	0xbe	N	N	N	1	L2D victim buffer is full
L2_FILLB_FULL	0xbf	N	N	N	1	L2D Fill buffer is full

11.8.4 L3 Cache Events

Table 11-24 summarizes the directly-measured L3 cache events. An extensive list of derived events is provided in Table 11-25.

Table 11-24. 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
L3_MISSES	0xdc	Y	Y	Y	1	L3 Misses
L3_READS	0xdd	Y	Y	Y	1	L3 Reads
L3_REFERENCES	0xdb	Y	Y	Y	1	L3 References
L3_WRITES	0xde	Y	Y	Y	1	L3 Writes

Table 11-25. 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	L2 Writeback Hits	L3_WRITES.L2_WB.HIT
L2_WB_MISSES	L2 Writeback Misses	L3_WRITES.L2_WB.MISS
L2_WB_REFERENCES	L2 Writeback References	L3_WRITES.L2_WB.ALL
L3_WRITE_HITS	L3 Write Hits	L3_WRITES.ALL.HIT

Table 11-25. Derived Monitors for L3 Unified Cache Events (Continued)

Symbol Name	Description	Equation
L3_WRITE_MISSES	L3 Write Misses	L3_WRITES.ALL.MISS
L3_WRITE_REFERENCES	L3 Write References	L3_WRITES.ALL.ALL

11.9 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 11-26. 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
EXTERN_DP_PINS_0_TO_3	0x9e	N	N	N	1	DP pins 0-3 asserted
EXTERN_DP_PINS_4_TO_5	0x9f	N	N	N	1	DP pins 4-5 asserted
SERIALIZATION_EVENTS	0x53	N	N	N	1	Number of srlz.I instructions

Table 11-27. Derived Monitors for System Events

Symbol Name	Description	Equation
CODE_DEBUG_REGISTER_MATCHES	Code Debug Register Matches	IA64_TAGGED_INST_RETIRED

11.10 TLB Events

The Itanium 2 processor instruction and data TLBs and the VHPT walker are monitored by the events described in Table 11-28

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. ITLB_INSERTS_HPW and DTLB_INSERTS_HPW count the number of instruction/data TLB inserts performed by the VHPT walker.

Table 11-28. 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 Installs to DTLB
DTLB_INSERTS_HPW_RETIRED	0x2c	Y	Y	Y	4	VHPT entries inserted into DTLB by the Hardware Page Walker
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 11-29. 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 Itanium 2 processor has 2 data TLBs called L1DTLB and L2DTLB (also referred to as DTLB or L2 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.

11.11 System Bus Events

Table 11-30 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, CPU clock cycles are inferred rather than bus clock cycles unless otherwise specified. Numerous derived events have been included in Table 11-31.

Table 11-30. Performance Monitors for System Bus Events

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
BUS_ALL	0x87	N	N	N	1	Bus Transactions
BUS_BRQ_LIVE_REQ_HI	0x9c	N	N	N	2	BRQ Live Requests (two most-significant-bit of the 5-bit outstanding BRQ request count)
BUS_BRQ_LIVE_REQ_LO	0x9b	N	N	N	7	BRQ Live Requests (three least-significant-bit of the 5-bit outstanding BRQ request count)
BUS_BRQ_REQ_INSERTED	0x9d	N	N	N	1	BRQ Requests Inserted
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
BUS_IOQ_LIVE_REQ_HI	0x98	N	N	N	2	In-order Bus Queue Requests (two most-significant-bit of the 4-bit outstanding IOQ request count)
BUS_IOQ_LIVE_REQ_LO	0x97	N	N	N	3	In-order Bus Queue Requests (two least-significant-bit of the 4-bit outstanding IOQ request count)
BUS_LOCK	0x93	N	N	N	1	IA-32 Compatible Bus Lock Transactions
BUS_BACKSNP_REQ	0x8e	N	N	N	1	Bus Back Snoop Requests
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

Table 11-30. Performance Monitors for System Bus Events (Continued)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
BUS_MEM_READ_OUT_HI	0x94	N	N	N	2	Outstanding memory RD transactions
BUS_MEM_READ_OUT_LO	0x95	N	N	N	7	Outstanding memory RD transactions
BUS_OOQ_LIVE_REQ_HI	0x9a	N	N	N	2	Out-of-order Bus Queue Requests (two most-significant-bit of the 4-bit outstanding OOQ request count)
BUS_OOQ_LIVE_REQ_LO	0x99	N	N	N	7	Out-of-order Bus Queue Requests (three least-significant-bit of the 4-bit outstanding OOQ request count)
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_ALL_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
BUS_SNOOPQ_REQ	0x96	N	N	N	7	Bus Snoop Queue Requests
BUS_SNOOPS	0x86	N	N	N	1	Bus Snoops Total
BUS_SNOOPS_HITM	0x85	N	N	N	1	Bus Snoops HIT Modified Cache Line
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

Table 11-31. Derived Monitors for System Bus Events

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	BUS_BRQ_LIVE_REQ_HI * 8 + BUS_BRQ_LIVE_REQ_LO

Table 11-31. Derived Monitors for System Bus Events (Continued)

Symbol Name	Description	Equation
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	BUS_IOQ_LIVE_REQ_HI * 4 + BUS_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	BUS_MEM_READ_OUT_HI * 8 + BUS_MEM_READ_OUT_LO
BUS_OOQ_LIVE_REQ	Out-of-order Bus Queue Requests	BUS_OOQ_LIVE_REQ_HI * 8 + BUS_OOQ_LIVE_REQ_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
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
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_BST_HITM	Bus Read Invalidate Line in Burst transactions (BRIL) resulting in HITMs	BUS_RD_INVAL_ALL_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

Table 11-32 defines the conventions that will be used when describing the Itanium 2 processor system bus transaction monitors in this section as well as the individual monitor descriptions in Section 11.14, “Performance Monitor Event List”.

Table 11-32. Conventions for System Bus Transactions

Name	Description
BRC	Memory Read Current (128 byte transactions). Reads without changing state.
BRL	Memory Read (64 byte bursts). Includes code fetches and data loads from WB memory.
BRIL	Memory Read & Invalidate (64 byte bursts). Also known as read for ownership (RFO).
BIL	Memory Read & Invalidate (0 byte sized transaction). Caused by flush cache (`fc`) instruction only.
BWL	Memory Write (64 byte bursts). Explicit writebacks/coalesced writes.
BRP	Partial Memory Reads (<64 byte transactions). Typically, uncacheable reads.
BWP	Partial Memory Write (<64 byte transactions). Typically, uncacheable writes.
IORD	Partial IO Read (<64 byte transactions). Uncacheable read to IO port space.
IOWR	Partial IO Write (<64 byte transactions). Uncacheable write to IO port space.

Other transactions besides those listed in Table 11-32 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 Itanium 2 processor provides local processor and remote response monitors. The local processor snoop events (BUS_SNOOPS_HITM, BUS_SNOOPS and BUS_SNOOPQ_REQ) monitor inbound snoop traffic. The remote response events (BUS_RD_HIT, BUS_RD_HITM, BUS_RD_INVAL_HITM and BUS_RD_INVAL_ALL_HITM) monitor the snoop responses of other processors to bus transactions that the monitoring processor originated. Table 11-33 summarizes the remote snoop events by bus transaction.

Table 11-33. Bus Events by Snoop Response

Remote Processor Response	BRL	BIL	BRIL
HIT	BUS_RD_HIT	n/a	n/a
HITM	BUS_RD_HITM	BUS_RD_INVAL_HITM	BUS_RD_INVAL_BST_HITM
ALL	BUS_RD_ALL	BUS_RD_INVAL	BUS_RD_INVAL

11.12 RSE Events

Register Stack Engine events are presented in Table 11-34. The number of current/dirty registers are split among three monitors since there are 96 physical registers in the Itanium 2 processor.

Table 11-34. Performance Monitors for RSE Events

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 11-34. Performance Monitors for RSE Events (Continued)

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 11-35. 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

11.13 Performance Monitors Ordered by Event Code

Table 11-36 presents all of the performance monitors provided in the Itanium 2 processor ordered by their event code.

Table 11-36. All Performance Monitors Ordered by Code

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_RSE_BUBBLE	0x01	N	N	N	1	Full pipe bubbles in main pipe due to RSE stalls
BE_EXE_BUBBLE	0x02	N	N	N	1	Full pipe bubbles in main pipe due to Execution unit stalls
FP_TRUE_SIRSTALL	0x03	Y	N	N	1	SIR stall asserted and leads to a trap

Table 11-36. All Performance Monitors Ordered by Code (Continued)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
BE_FLUSH_BUBBLE	0x04	N	N	N	1	Full pipe bubbles in main pipe due to flushes
FP_FALSE_SIRSTALL	0x05	Y	N	N	1	SIR stall without a trap
FP_FAILED_FCHKF	0x06	Y	N	N	1	Failed fchkf
IA32_ISA_TRANSITIONS	0x07	N	N	N	1	Itanium to/from IA-32 ISA Transitions
IA64_INST_RETIRED	0x08	Y	N	Y	6	Retired Itanium Instructions
IA64_TAGGED_INST_RETIRED	0x08	Y	N	Y	6	Retired Tagged Instructions
FP_OPS_RETIRED	0x09	Y	N	N	4	Retired FP operations
FP_FLUSH_TO_ZERO	0x0b	Y	N	N	2	FP Result Flushed to Zero
BRANCH_EVENT	0x11	Y	N	Y	1	Branch Event Captured
CPU_CYCLES	0x12	N	N	N	1	CPU Cycles
CPU_CPL_CHANGES	0x13	N	N	N	1	Privilege Level Changes
RSE_REFERENCES_RETIRED	0x20	Y	Y	Y	2	RSE Accesses
RSE_DIRTY_REGS_6	0x24	N	N	N	1	Dirty RSE registers
RSE_CURRENT_REGS_6	0x26	N	N	N	1	Current RSE registers
RSE_DIRTY_REGS_5_TO_3	0x28	N	N	N	7	Dirty RSE registers
RSE_DIRTY_REGS_2_TO_0	0x29	N	N	N	7	Dirty RSE registers
RSE_CURRENT_REGS_5_TO_3	0x2a	N	N	N	7	Current RSE registers
RSE_CURRENT_REGS_2_TO_0	0x2b	N	N	N	7	Current RSE registers
DTLB_INSERTS_HPW_RETIRED	0x2c	Y	Y	Y	4	VHPT entries inserted into DTLB by HW PW
HPW_DATA_REFERENCES	0x2d	Y	Y	Y	4	Data memory references to VHPT
RSE_EVENT_RETIRED	0x32	N	N	N	1	Retired RSE operations
L1I_READS	0x40	Y	N	N	1	L1 Instruction Cache Reads
L1I_FILLS	0x41	Y	N	N	1	L1 Instruction Cache Fills
L2_INST_DEMAND_READS	0x42	Y	N	N	1	L1 Instruction Cache and ISB Misses
L1I_EAR_EVENTS	0x43	Y	N	N	1	Instruction EAR Events
L1I_PREFETCHES	0x44	Y	N	N	1	L1 Instruction Prefetch Requests
L2_INST_PREFETCHES	0x45	Y	N	N	1	L2 Instruction Prefetch Requests
ISB_BUNPAIRS_IN	0x46	Y	N	N	1	Bundle pairs written from L2 into FE
ITLB_MISSES_FETCH	0x47	Y	N	N	1	ITLB Misses Demand Fetch
L1ITLB_INSERTS_HPW	0x48	Y	N	N	1	L1ITLB Hardware Page Walker Inserts
DISP_STALLED	0x49	N	N	N	1	Number of cycles dispersal stalled
L1I_SNOOP	0x4a	Y	Y	Y	1	Snoop requests handled by L1I
L1I_PURGE	0x4b	Y	N	N	1	L1ITLB purges handled by L1I
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

Table 11-36. All Performance Monitors Ordered by Code (Continued)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
SYLL_OVERCOUNT	0x4f	Y	N	N	2	Syllables overcounted
NOPS_RETIRED	0x50	Y	N	Y	6	Retired NOP Instructions
PREDICATE_SQUASHED_RETIRED	0x51	Y	N	Y	6	Instructions Squashed Due to Predicate Off
DATA_DEBUG_REGISTER_FAULT	0x52	N	N	N	1	Fault due to data debug reg. Match to load/store instruction
SERIALIZATION_EVENTS	0x53	N	N	N	1	Number of srlz.I instructions
BR_PATH_PRED	0x54	Y	N	Y	3	FE Branch Path Prediction Detail
INST_FAILED_CHKS_RETIRED	0x55	N	N	N	1	Failed Speculative Check Loads
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
ALAT_CAPACITY_MISS	0x58	Y	Y	Y	2	ALAT Entry Replaced
IA32_INST_RETIRED	0x59	N	N	N	2	IA-32 Instructions Retired
BR_MISPRED_DETAIL	0x5b	Y	N	Y	3	FE Branch Mispredict Detail
L1I_STRM_PREFETCHES	0x5f	Y	N	N	1	L1 Instruction Cache line prefetch requests
L1I_RAB_FULL	0x60	N	N	N	1	Is RAB full?
BE_BR_MISPRED_DETAIL	0x61	Y	N	Y	1	BE branch misprediction detail
ENCBR_MISPRED_DETAIL	0x63	Y	N	Y	1	Number of encoded branches retired
L1I_RAB_ALMOST_FULL	0x64	N	N	N	1	Is RAB almost full?
L1I_FETCH_RAB_HIT	0x65	Y	N	N	1	Instruction fetch hitting in RAB
L1I_FETCH_ISB_HIT	0x66	Y	N	N	1	“Just-in-time” instruction fetch hitting in and being bypassed from ISB
L1I_PREFETCH_STALL	0x67	N	N	N	1	Why prefetch pipeline is stalled?
BR_MISPRED_DETAIL2	0x68	Y	N	Y	2	FE Branch Mispredict Detail (Unknown path component)
L1I_PVAB_OVERFLOW	0x69	N	N	N	1	PVAB overflow
BR_PATH_PRED2	0x6a	Y	N	Y	2	FE Branch Path Prediction Detail (Unknown prediction component)
FE_LOST_BW	0x70	N	N	N	2	Invalid bundles at the entrance to IB
FE_BUBBLE	0x71	N	N	N	1	Bubbles seen by FE
BE_LOST_BW_DUE_TO_FE	0x72	N	N	N	2	Invalid bundles if BE not stalled for other reasons
IDEAL_BE_LOST_BW_DUE_TO_FE	0x73	N	N	N	2	Invalid bundles at the exit from IB
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_HITM	0x82	N	N	N	1	Bus BIL Transaction Results in HITM
BUS_RD_INVAL_ALL_HITM	0x83	N	N	N	1	Bus BIL or BRIL Transaction Results in HITM

Table 11-36. All Performance Monitors Ordered by Code (Continued)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
BUS_HITM	0x84	N	N	N	1	Bus Hit Modified Line Transactions
BUS_SNOOPS_HITM	0x85	N	N	N	1	Bus Snoops HIT Modified Cache Line
BUS_SNOOPS	0x86	N	N	N	1	Bus Snoops Total
BUS_ALL	0x87	N	N	N	1	Bus Transactions
BUS_DATA_CYCLE	0x88	N	N	N	1	Valid data cycle on the Bus
MEM_READ_CURRENT	0x89	N	N	N	1	Current Mem Read Transactions On Bus
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
BUS_RD_DATA	0x8c	N	N	N	1	Bus Read Data Transactions
BUS_RD_PRTL	0x8d	N	N	N	1	Bus Read Partial Transactions
BUS_BACKSNP_REQ	0x8e	N	N	N	1	Bus Back Snoop Requests
BUS_SNOOP_STALL_CYCLES	0x8f	N	N	N	1	Bus Snoop Stall Cycles (from any agent)
BUS_IO	0x90	N	N	N	1	IA-32 Compatible IO Bus Transactions
BUS_RD_IO	0x91	N	N	N	1	IA-32 Compatible IO Read Transactions
BUS_WR_WB	0x92	N	N	N	1	Bus Write Back Transactions
BUS_LOCK	0x93	N	N	N	1	IA-32 Compatible Bus Lock Transactions
BUS_MEM_READ_OUT_HI	0x94	N	N	N	2	Outstanding memory RD transactions
BUS_MEM_READ_OUT_LO	0x95	N	N	N	7	Outstanding memory RD transactions
BUS_SNOOPQ_REQ	0x96	N	N	N	7	Bus Snoop Queue Requests
BUS_IOQ_LIVE_REQ_LO	0x97	N	N	N	3	Inorder Bus Queue Requests (two least-significant-bit of the 4-bit outstanding IOQ request count)
BUS_IOQ_LIVE_REQ_HI	0x98	N	N	N	2	Inorder Bus Queue Requests (two most-significant-bit of the 4-bit outstanding IOQ request count)
BUS_OOQ_LIVE_REQ_LO	0x99	N	N	N	7	Out-of-order Bus Queue Requests (three least-significant-bit of the 4-bit outstanding OOQ request count)
BUS_OOQ_LIVE_REQ_HI	0x9a	N	N	N	2	Out-of-order Bus Queue Requests (two most-significant-bit of the 4-bit outstanding OOQ request count)
BUS_BRQ_LIVE_REQ_LO	0x9b	N	N	N	7	BRQ Live Requests (three least-significant-bit of the 5-bit outstanding BRQ request count)
BUS_BRQ_LIVE_REQ_HI	0x9c	N	N	N	2	BRQ Live Requests (two most-significant-bit of the 5-bit outstanding BRQ request count)
BUS_BRQ_REQ_INSERTED	0x9d	N	N	N	1	BRQ Requests Inserted
EXTERN_DP_PINS_0_TO_3	0x9e	N	N	N	1	DP pins 0-3 asserted

Table 11-36. All Performance Monitors Ordered by Code (Continued)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
EXTERN_DP_PINS_4_TO_5	0x9f	N	N	N	1	DP pins 4-5 asserted
L2_OZQ_CANCELS0	0xa0	Y	Y	Y	4	L2 OZQ cancels
L2_IFET_CANCELS	0xa1,0xa5,0xa9,0xad	Y	Y	Y	1	Instruction fetch cancels by the L2.
L2_OZQ_ACQUIRE	0xa2,0xa6,0xaa,0xae	N	N	N	1	Clocks with acquire ordering attribute existed in L2 OZQ
L2_OZQ_RELEASE	0xa3,0xa7,0xab,0xaf	N	N	N	1	Clocks with release ordering attribute existed in L2 OZQ
L2_OZQ_CANCELS2	0xa8	Y	Y	Y	4	L2 OZQ cancels
L2_OZQ_CANCELS1	0xac	Y	Y	Y	4	L2 OZQ cancels
L2_L3ACCESS_CANCEL	0xb0	Y	Y	Y	1	Canceled L3 accesses
L2_REFERENCES	0xb1	Y	Y	Y	4	Requests made from L2
L2_DATA_REFERENCES	0xb2	Y	Y	Y	4	Data read/write access to L2
L2_FORCE_RECIRC	0xb4	Y	Y	Y	4	Forced recirculates
L2_ISSUED_RECIRC_OZQ_ACC	0xb5	Y	Y	Y	1	Count number of times a recirculate issue was attempted and not preempted
L2_GOT_RECIRC_OZQ_ACC	0xb6	Y	Y	Y	1	Counts number of OZQ accesses recirculated back to L1D
L2_SYNTH_PROBE	0xb7	Y	Y	Y	1	Synthesized Probe
L2_BYPASS	0xb8	Y	Y	Y	1	Count bypass
L2_OPS_ISSUED	0xb8	Y	Y	Y	4	Different operations issued by L2D
L2_ISSUED_RECIRC_IFETCH	0xb9	Y	Y	Y	1	Instruction fetch recirculates issued by L2D
L2_BAD_LINES_SELECTED	0xb9	Y	Y	Y	4	Valid line replaced when invalid line is available
L2_GOT_RECIRC_IFETCH	0xba	Y	Y	Y	1	Instruction fetch recirculates received by L2D
L2_STORE_HIT_SHARED	0xba	Y	Y	Y	2	Store hit a shared line
TAGGED_L2_DATA_RETURN_PORT	0xbb	Y	Y	Y	1	Tagged L2 Data Return Ports 0/1
L2_OZQ_FULL	0xbc	N	N	N	1	L2D OZQ is full
L2_OZDB_FULL	0xbd	N	N	N	1	L2D OZ data buffer is full
L2_VICTIMB_FULL	0xbe	N	N	N	1	L2D victim buffer is full
L2_FILLB_FULL	0xbf	N	N	N	1	L2D Fill buffer is full
L1DTLB_TRANSFER	0xc0	Y	Y	Y	1	L1DTLB misses that hit in the L2DTLB for accesses counted in L1D_READS
L2DTLB_MISSES	0xc1	Y	Y	Y	4	L2DTLB Misses

Table 11-36. All Performance Monitors Ordered by Code (Continued)

Symbol Name	Event Code	IAR	DAR	OPC	Max Inc/Cyc	Description
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
DATA_DEBUG_REGISTER_MATCHES	0xc6	Y	Y	Y	1	Data debug register matches data address of memory reference
L1D_READ_MISSES	0xc7	Y	Y	Y	2	L1 Data Cache Read Misses
DATA_EAR_EVENTS	0xc8	Y	Y	Y	1	L1 Data Cache EAR Events
DTLB_INSERTS_HPW	0xc9	Y	Y	Y	4	Hardware Page Walker Installs to DTLB
BE_L1D_FPU_BUBBLE	0xca	N	N	N	1	Full pipe bubbles in main pipe due to FP or L1 dcache
L2_MISSES	0xcb	Y	Y	Y	1	L2 Misses
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
UC_STORES_RETIRED	0xd0	Y	Y	Y	2	Retired Uncacheable Stores
STORES_RETIRED	0xd1	Y	Y	Y	2	Retired Stores
MISALIGNED_STORES_RETIRED	0xd2	Y	Y	Y	2	Retired Misaligned Store Instructions
L3_REFERENCES	0xdb	Y	Y	Y	1	L3 References
L3_MISSES	0xdc	Y	Y	Y	1	L3 Misses
L3_READS	0xdd	Y	Y	Y	1	L3 Reads
L3_WRITES	0xde	Y	Y	Y	1	L3 Writes
L3_LINES_REPLACED	0xdf	N	N	N	1	L3 Cache Lines Replaced

11.14 Performance Monitor Event List

This section enumerates Itanium 2 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
- **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 11-37. 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
- **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.

Table 11-38. 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
- **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 11-39. 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
- **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 11-40. 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
- **Definition:** Counts the number of full-pipe bubbles in the main pipe due to flushes.
- **NOTE:** XPN is higher priority than BRU.

Table 11-41. 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
XPN	bxx10	Back-end was stalled due to an exception/interruption flush
---	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
- **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 11-42. Unit Masks for BE_L1D_FPU_BUBBLE

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_DCURECIR	b0100	Back-end was stalled by L1D due to DCU recirculating
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_DCS	b1000	Back-end was stalled by L1D due to dcs requiring a stall
L1D_L2BPRESS	b1001	Back-end was stalled by L1D due to L2 Back Pressure

Table 11-42. Unit Masks for BE_L1D_FPU_BUBBLE (Continued)

Extension	PMC.umask [19:16]	Description
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.
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
- **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 11-43. 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	(* count is undefined *)
—	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

Table 11-43. Unit Masks for BE_LOST_BW_DUE_TO_FE (Continued)

Extension	PMC.umask [19:16]	Description
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
- **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 11-44. 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 *)

BRANCH_EVENT

- **Title:** Branch Event Captured
- **Category:** Branch Events **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x11, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of branch bundles retired which match the constraints of PMC12 (defined under “Performance Monitor Control Registers”).

BR_MISPRED_DETAIL

- **Title:** FE Branch Mispredict Detail
- **Category:** Branch Events **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x5b, **Max. Inc/Cyc:** 3
- **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 11-45. 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
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
- **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 11-46. Unit Masks for BR_MISPREDICT_DETAIL2

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)
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
- **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 11-47. Unit Masks for BR_PATH_PRED

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
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
- **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 11-48. Unit Masks for BR_PATH_PRED2

Extension	PMC.umask [19:16]	Description
ALL.UNKNOWNPRED_NOT TAKEN	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)
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:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x87, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of bus transactions.

Table 11-49. Unit Masks for BUS_ALL

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
IO	bxx01	Non-CPU priority agents
SELF	bxx10	Local processor
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_BACKSNP_REQ

- **Title:** Bus Back Snoop Requests
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x8e, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of bus back snoop me requests accepted by the bus unit.

Table 11-50. Unit Masks for BUS_BACKSNP_REQ

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
THIS	bxx01	Counts the number of bus back snoop me requests
—	bxx10	(* nothing will be counted *)
—	bxx11	(* nothing will be counted *)

BUS_BRQ_LIVE_REQ_HI

- **Title:** BRQ Live Requests (upper two bits)
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x9c, **Max. Inc/Cyc:** 2
- **Definition:** Counts the number of live read requests in BRQ. The Itanium 2 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 a 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 a victim, the life of read in BRQ is extended).

BUS_BRQ_LIVE_REQ_LO

- **Title:** BRQ Live Requests (lower three bits)
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x9b, **Max. Inc/Cyc:** 7
- **Definition:** Counts the number of live read requests in BRQ. The Itanium 2 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 a 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 a victim, the life of read in BRQ is extended).

BUS_BRQ_REQ_INSERTED

- **Title:** BRQ Requests Inserted
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x9d, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of requests which are inserted into BRQ.
- **NOTE:** Entries made into BRQ due to L2 victims (caused by read, fc, cc) are not counted.

BUS_DATA_CYCLE

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

BUS_HITM

- **Title:** Bus Hit Modified Line Transactions
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x84, **Max. Inc/Cyc:** 1
- **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_ALL_HITM + BUS_RD_HITM.

BUS_IO

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

Table 11-51. Unit Masks for BUS_IO

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
IO	bxx01	Non-CPU priority agents
SELF	bxx10	Local processor
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_IOQ_LIVE_REQ_HI

- **Title:** Inorder Bus Queue Requests (upper two bits)
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x98, **Max. Inc/Cyc:** 2
- **Definition:** Counts the number of live in-order bus requests. The Itanium 2 processor can have a total of 8 per cycle. The upper two bits are stored in this counter.

BUS_IOQ_LIVE_REQ_LO

- **Title:** Inorder Bus Queue Requests (lower two bits)
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x97, **Max. Inc/Cyc:** 3
- **Definition:** Counts the number of live in-order bus requests. The Itanium 2 processor can have a total of 8 per cycle. The lower two bits are stored in this counter.

BUS_LOCK

- **Title:** IA-32 Compatible Bus Lock Transactions
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x93, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of IA-32 bus lock transactions.

Table 11-52. Unit Masks for BUS_LOCK

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
—	bxx01	(* illegal selection *)
SELF	bxx10	Local processor
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_MEMORY

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

Table 11-53. Unit Masks for BUS_MEMORY

Extension	PMC.umask [19:16]	Description
—	b00xx	(* nothing will be counted *)
—	b0100	(* nothing will be counted *)
EQ_128BYTE.IO	b0101	Number of full cache line transactions (BRL, BRIL, BWL) from non-CPU priority agents
EQ_128BYTE.SELF	b0110	Number of full cache line transactions (BRL, BRIL, BWL) from local processor
EQ_128BYTE.ANY	b0111	Number of full cache line transactions (BRL, BRIL, BWL) from CPU or non-CPU (all transactions).
—	b1000	(* nothing will be counted *)
LT_128BYTE.IO	b1001	Number of less than full cache line transactions (BRP, BWP) from non-CPU priority agents
LT_128BYTE.SELF	b1010	Number of less than full cache line transactions (BRP, BWP) local processor

Table 11-53. Unit Masks for BUS_MEMORY (Continued)

Extension	PMC.umask [19:16]	Description
LT_128BYTE.ANY	b1011	Number of less than full cache line transactions (BRP, BWP) CPU or non-CPU (all transactions).
—	b1100	(* nothing will be counted *)
ALL.IO	b1101	All bus transactions from non-CPU priority agents
ALL.SELF	b1110	All bus transactions from 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:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x8b, **Max. Inc/Cyc:** 1
- **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 11-54. Unit Masks for BUS_MEM_READ

Extension	PMC.umask [19:16]	Description
—	b0000	(* nothing will be counted *)
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 local processor
BIL.ANY	b0011	Number of BIL 0-byte memory read invalidate transactions from CPU or non-CPU (all transactions).
—	b0100	(* nothing will be counted *)
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 local processor
BRL.ANY	b0111	Number of full cache line memory read transactions from CPU or non-CPU (all transactions).
---	b1000	(* nothing will be counted *)
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 local processor
BRIL.ANY	b1011	Number of full cache line memory read invalidate transactions from CPU or non-CPU (all transactions).
—	b1100	(* nothing will be counted *)
ALL.IO	b1101	All memory read transactions from non-CPU priority agents

Table 11-54. Unit Masks for BUS_MEM_READ (Continued)

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

BUS_MEM_READ_OUT_HI

- **Title:** Outstanding Memory Read Transactions (upper 2 bits)
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x94, **Max. Inc/Cyc:** 2
- **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 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 L2.
- **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 [all,self] for average system memory latency.

BUS_MEM_READ_OUT_LO

- **Title:** Outstanding Memory Read Transactions (lower 3 bits)
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x95, **Max. Inc/Cyc:** 7
- **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 L2.
- **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 [all,self] for average system memory latency.

BUS_OOQ_LIVE_REQ_HI

- **Title:** Out-of-order Bus Queue Requests (upper 2 bits)
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x9a, **Max. Inc/Cyc:** 2
- **Definition:** Counts the number of live deferred (out-of-order) bus requests. The Itanium 2 processor can have a total of 18 of this event per cycle. The upper two bits are stored in this counter (bits 4:3). This event increments every CPU clock cycle. The counter is incremented by the number of live deferred transactions at that time.
- **NOTE:** BUS_OOO_LIVE_REQ/CPU_CYCLES event will be an indication of average number of outstanding deferred transactions per CPU clock.

BUS_OOQ_LIVE_REQ_LO

- **Title:** Out-of-order Bus Queue Requests (lower 3 bits)
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x99, **Max. Inc/Cyc:** 7
- **Definition:** Counts the number of live deferred (out-of-order) bus requests. The Itanium 2 processor can have a total of 18 of this event per cycle. The lower three bits are stored in this

counter (bits 2:0). This event increments every CPU clock cycle. The counter is incremented by the number of live deferred transactions at that time.

- **NOTE:** BUS_OOO_LIVE_REQ/CPU_CYCLES event will be an indication of average number of outstanding deferred transactions per CPU clock.

BUS_RD_DATA

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

Table 11-55. Unit Masks for BUS_RD_DATA

Extension	PMC.umask [19:16]	Description
---	bxx00	(* nothing will be counted *)
IO	bxx01	Non-CPU priority agents
SELF	bxx10	Local processor
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_RD_HIT

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

BUS_RD_HITM

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

BUS_RD_INVAL_ALL_HITM

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

BUS_RD_INVAL_HITM

- **Title:** Bus BIL Transaction Results in HITM
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x82, **Max. Inc/Cyc:** 1
- **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.

BUS_RD_IO

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

Table 11-56. Unit Masks for BUS_RD_IO

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
IO	bxx01	Non-CPU priority agents
SELF	bxx10	Local processor
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_RD_PRTL

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

Table 11-57. Unit Masks for BUS_RD_PRTL

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
IO	bxx01	Non-CPU priority agents
SELF	bxx10	Local processor
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_SNOOPQ_REQ

- **Title:** Bus Snoop Queue Requests
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x96, **Max. Inc/Cyc:** 7
- **Definition:** Counts the number of live snoop responses. This event increments every CPU clock cycle. The amount that counter is incremented is the number of outstanding snoop responses at that time.

BUS_SNOOPS

- **Title:** Bus Snoops Total
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x86, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of bus snoop requests on the bus.

Table 11-58. Unit Masks for BUS_SNOOPS

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
IO	bxx01	Non-CPU priority agents
SELF	bxx10	Local processor
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_SNOOPS_HITM

- **Title:** Bus Snoops HIT Modified Cache Line
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x85, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of bus snoop requests from remote processors that hit a modified line in the local processor.

Table 11-59. Unit Masks for BUS_SNOOPS_HITM

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
—	bxx01	(* illegal selection *)
SELF	bxx10	Local processor
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_SNOOP_STALL_CYCLES

- **Title:** Bus Snoop Stall Cycles (from any agent)
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x8f, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of bus clocks FSB is stalled for snoop (this is twice the number of bus clocks HIT and HITM are asserted at the same time).

Table 11-60. Unit Masks for BUS_SNOOP_STALL_CYCLES

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
—	bxx01	(* illegal selection *)
SELF	bxx10	Local processor
ANY	bxx11	CPU or non-CPU (all transactions).

BUS_WR_WB

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

Table 11-61. Unit Masks for BUS_WR_WB

Extension	PMC.umask [19:16]	Description
—	b00xx	(* nothing will be counted *)
—	b0100	(* nothing 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	Local 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.
—	b1000	(* nothing will be counted *)
—	b1001	(* illegal selection *)
CCASTOUT.SELF	b1010	Local 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
—	b1100	(* nothing will be counted *)
ALL.IO	b1101	Non-CPU priority agents
ALL.SELF	b1110	Local 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
- **Definition:** Counts the number of privilege level changes.

CPU_CYCLES

- **Title:** CPU Cycles
- **Category:** Basic Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x12, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of clock cycles.

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
- **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
- **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
- **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
- **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
- **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
- **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:** Y/Y/Y
- **Event Code:** 0xc9, **Max. Inc/Cyc:** 4
- **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.

DTLB_INSERTS_HPW_RETIRED

- **Title:** VHPT Entries Inserted into DTLB by the Hardware Page Walker
- **Category:** TLB **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0x2c, **Max. Inc/Cyc:** 4
- **Definition:** Counts the number of VHPT entries inserted into DTLB by Hardware Page Walker
- **NOTE:** This will not include misses which the DTLB did not squash even though the instructions causing the miss did not get to retirement. The difference between this event and DTLB_INSERTS_HPW is the amount of potentially unnecessary inserts into DTLB.

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
- **Definition:** Counts the number of branches retired only if there is a branch on port B0 (i.e. encoded branch).

Table 11-62. 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_ ARGET	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

EXTERN_DP_PINS_0_TO_3

- **Title:** DP Pins 0-3 Asserted
- **Category:** System Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x9e, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of bus clocks external DP pins 0 through 3 were asserted.

Table 11-63. Unit Masks for EXTERN_DP_PINS_0_TO_3

Extension	PMC.umask [19:16]	Description
—	b0000	(* nothing will be counted *)
PIN0	bxxx1	Include pin0 assertion
PIN1	bxx1x	Include pin1 assertion
PIN2	bx1xx	Include pin2 assertion
PIN3	b1xxx	Include pin3 assertion

EXTERN_DP_PINS_4_TO_5

- **Title:** DP Pins 4-5 Asserted
- **Category:** System Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x9f, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of bus clocks external DP pins 4 and 5 were asserted.

Table 11-64. Unit Masks for EXTERN_DP_PINS_4_TO_5

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
PIN4	bxxx1	Include pin4 assertion
PIN5	bxx1x	Include pin5 assertion

FE_BUBBLE

- **Title:** Bubbles Seen by FE
- **Category:** Stall Events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x71, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of bubbles seen by front-end. This event is another way of looking at the FE_LOST_BW event.
- **NOTE:** 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 11-65. Unit Masks for FE_BUBBLE

Extension	PMC.umask [19:16]	Description
ALL	b0000	Count regardless of cause
FEFLUSH	b0001	Only if caused by a front-end flush
—	b0010	(* count is undefined *)
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
BUBBLE	b1101	Only if caused by branch bubble stall
—	b1110-b1111	(* nothing will be counted *)

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
- **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 11-66. 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	(* count is undefined *)
—	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
- **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
- **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
- **Definition:** Counts the number of times a near zero result is flushed to zero in FTZ mode.

FP_OPS_RETIRED

- **Title:** Retired FP Operations
- **Category:** Instruction Execution **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x09, **Max. Inc/Cyc:** 4
- **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
- **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:** Y/Y/Y
- **Event Code:** 0x2d, **Max. Inc/Cyc:** 4
- **Definition:** Counts the number of data memory references to VHPT.
- **NOTE:** If HPW is enabled all the time, this event and L2DTLB_MISSES are equivalent. If HPW is disabled all the time, this event should count 0. This will include misses the L2DTLB did not squash even though the instructions causing the miss did not get to retirement.

IA32_INST_RETIRED

- **Title:** IA-32 Instructions Retired
- **Category:** Basic events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x59, **Max. Inc/Cyc:** 2
- **Definition:** Counts the number of IA-32 instructions retired.

IA32_ISA_TRANSITIONS

- **Title:** Itanium to/from IA-32 ISA Transitions
- **Category:** Basic events **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x07, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of times instruction set transitions from Itanium to IA-32 or from IA-32 to Itanium (Number of times PSR.is bit toggles).

IA64_INST_RETIRED

- **Title:** Retired Itanium Instructions
- **Category:** Basic Events **IAR/DAR/OPC:** Y/N/Y
- **Event Code:** 0x08, **Max. Inc/Cyc:** 6
- **Definition:** Counts the number of retired instructions excluding hardware generated RSE operations and instructions that were predicated off. This event includes all non-branch instructions which reached retirement with a true predicate and all branches regardless of predicate. 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).

  An example of non-default setup follows:

  Let's say we want to use IBRP2-PMC8 for measuring IA64_INST_RETIRED in PMD4. The following bits need to be programmed to make this happen.

  PMC4.umask = xx10

  PMC14.IBRP2 = 1 (PMC14 is also known as IPF_IBRC)

  PMC15.IBRP2_PMC8 = 1 (PMC15 is also known as ISD_DEBUGTAG). Note that PMC8 can still be used for the IBRP0_PMC8 umask.

Table 11-67. 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
- **Definition:** Counts the number of retired instructions, excluding hardware generated RSE operations and instructions that were predicated off, that match the Instruction Address Breakpoint (IBRs) and Opcode Match register settings (PMC8,9). This event includes all non-branch instructions which reached retirement with a true predicate and all branches regardless of predicate. See Section 10.3.5, "Instruction Address Range Matching" for more details about how to program different registers.
- **NOTE:** MLX bundles will be counted as no more than two instructions.

Table 11-68. Unit Masks for IA64_TAGGED_INST_RETIRED

Extension	PMC.umask [19:16]	Description
IBRP0_PMC8	bxx00	Instruction tagged by Instruction Breakpoint Pair 0 and opcode matcher PMC8. Code executed with PSR.is=1 is included.
IBRP1_PMC9	bxx01	Instruction tagged by Instruction Breakpoint Pair 1 and opcode matcher PMC9. Code executed with PSR.is=1 is included.
IBRP2_PMC8	bxx10	Instruction tagged by Instruction Breakpoint Pair 2 and opcode matcher PMC8. Code executed with PSR.is=1 is not included.
IBRP3_PMC9	bxx11	Instruction tagged by Instruction Breakpoint Pair 3 and opcode matcher PMC9. Code executed with PSR.is=1 is not included.

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
- **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 11-69. 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	(* count is undefined *)
—	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

Table 11-69. Unit Masks for IDEAL_BE_LOST_BW_DUE_TO_FE (Continued)

Extension	PMC.umask [19:16]	Description
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
- **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 11-70. 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
- **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
- **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 11-71. Unit Masks for INST_FAILED_CHKA_LDC_ALAT

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
INT	bxx01	Only integer instructions

Table 11-71. Unit Masks for INST_FAILED_CHKA_LDC_ALAT (Continued)

Extension	PMC.umask [19:16]	Description
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
- **Definition:** Provides information on the number of failed speculative check instructions (`chk.s`).

Table 11-72. 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 L2 into FE
- **Category:** L1 Instruction Cache and prefetch **IAR/DAR/OPC:** Y/N/N
- **Event Code:** 0x46, **Max. Inc/Cyc:** 1
- **Definition:** Provides information about the number of bundle pairs (32 bytes) written from L2 (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
- **Definition:** Counts the number of ITLB misses for demand fetch.

Table 11-73. 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.

Table 11-73. Unit Masks for ITLB_MISSES_FETCH (Continued)

Extension	PMC.umask [19:16]	Description
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
- **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.

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
- **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
- **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
- **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 11-74. 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
- **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
- **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
- **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 L2. It will count “just-in-time bypasses,” i.e. when instruction data is delivered by the L2 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
- **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
- **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
- **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
- **Definition:** Provides Information on why the prefetch pipeline is stalled.

Table 11-75. Unit Masks for L1I_PREFETCH_STALL

Extension	PMC.umask [19:16]	Description
—	bxx00-bxx01	(* nothing will be counted *)
FLOW	bxx10	Number of clocks flow is not asserted
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
- **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
- **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
- **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
- **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
- **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 L2_INST_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
- **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. Although each IA32 store will produce 1 snoop request to L1I, it will be counted here as many times as it is recirculated in L1D because it is busy doing more important things. 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
- **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.

L2_BAD_LINES_SELECTED

- **Title:** Valid Line Replaced When Invalid Line Is Available
- **Category:** L2 Unified Cache/L2 Cache Set 3 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb9, **Max. Inc/Cyc:** 4
- **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 3 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4. Shares Event Code with L2_ISSUED_RECIRC_IFETCH.

Table 11-76. Unit Masks for L2_BAD_LINES_SELECTED

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

L2_BYPASS

- **Title:** Count L2 Bypasses
- **Category:** L2 Unified Cache/L2 Cache Set 3 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb8, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of times a bypass occurred.
- **NOTE:** This is a restricted set 3 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4. Shares Event Code with L2_OPS_ISSUED.

Table 11-77. Unit Masks for L2_BYPASS

Extension	PMC.umask [19:16]	Description
L2_DATA1	b0000	Count only L2 data bypasses (L1D to L2A)
L2_DATA2	b0001	Count only L2 data bypasses (L1W to L2I)
L3_DATA1	b0010	Count only L3 data bypasses (L1D to L2A)

Table 11-77. Unit Masks for L2_BYPASS (Continued)

Extension	PMC.umask [19:16]	Description
—	b0011	(* nothing will be counted *)
L2_INST1	b0100	Count only L2 instruction bypasses (L1D to L2A)
L2_INST2	b0101	Count only L2 instruction bypasses (L1W to L2I)
L3_INST1	b0110	Count only L3 instruction bypasses (L1D to L2A)
—	b0111	(* nothing will be counted *)

L2_DATA_REFERENCES

- **Title:** Data Read/Write Access to L2
- **Category:** L2 Unified Cache/L2 Cache Set 1 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb2, **Max. Inc/Cyc:** 4
- **Definition:** Counts the number of requests made to L2 due to a data read and/or write accesses. The reported count is the number of requests prior to cache line merging. Semaphore operations are counted as one read and one write.
- **NOTE:** This is a restricted set 1 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4.

Table 11-78. Unit Masks for L2_DATA_REFERENCES

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
L2_DATA_READS	bxx01	Count only data read and semaphore operations.
L2_DATA_WRITES	bxx10	Count only data write and semaphore operations
L2_ALL	bxx11	Count both read and write operations (semaphores will count as 2)

L2DTLB_MISSES

- **Title:** L2DTLB Misses
- **Category:** TLB/L1D Cache Set 0 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xc1, **Max. Inc/Cyc:** 4
- **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.

L2_FILLB_FULL

- **Title:** L2 Fill Buffer Is Full
- **Category:** L2 Unified Cache **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xbf, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of times L2 Fill Buffer is full.
- **NOTE:** This is a restricted set 5 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4.

Table 11-79. Unit Masks for L2_FILLB_FULL

Extension	PMC.umask [19:16]	Description
THIS	b0000	L2 Fill buffer is full
—	b0001-b1111	(* count is undefined *)

L2_FORCE_RECIRC

- **Title:** Forced Recirculates
- **Category:** L2 Unified Cache/L2 Cache Set 2 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb4, **Max. Inc/Cyc:** 4
- **Definition:** Counts the number of L2 ops forced to recirculate, with the exception of SNP_OR_L3. SNP_OR_L3 will measure the number of times L2 ops are forced to recirculate. Anywhere from 0-32 ops can be affected by this one. All categories with the exception of SMC_HIT, TRAN_PERF, and SNP_OR_L3 occur at the insertion into the OZQ. SMC_HIT is when an ifetch is about to be written into the IPFQ and is forced to recirculate because there is an outstanding store to the same address. SNP_OR_L3 is when an existing OZQ entry is forced to recirculate because an incoming 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.
- **NOTE:** This is a restricted set 2 L2 Cache event. This event must be measured by PMD4.

Table 11-80. Unit Masks for L2_FORCE_RECIRC

Extension	PMC.umask [19:16]	Description
ANY	b0000	Count forced recirculates regardless of cause. SMC_HIT, TRAN_PREF & SNP_OR_L3 will not be included here.
SMC_HIT	b0001	Count only those caused by SMC hits due to an ifetch and load to same cache line or a pending WT store
L1W	b0010	Count only those caused by forced limbo
—	b0011	(* nothing will be counted *)
TAG_NOTOK	b0100	Count only those caused by L2 hits caused by in flight snoops, stores with a sibling miss to the same index, sibling probe to the same line or a pending sync.ia instruction
TRAN_PREF	b0101	Count only those caused by transforms to prefetches
SNP_OR_L3	b0110	Count only those caused by a snoop or L3 issue
—	b0111	(* nothing will be counted *)
VIC_PEND	b1000	Count only those caused by an L2 miss with pending victim

Table 11-80. Unit Masks for L2_FORCE_RECIRC (Continued)

Extension	PMC.umask [19:16]	Description
FILL_HIT	b1001	Count only those caused by an L2 miss which hit in the fill buffer.
IPF_MISS	b1010	Caused by L2 miss when instruction prefetch buffer miss already existed
VIC_BUF_FULL	b1011	Count only those caused by an L2 miss with victim buffer full
OZQ_MISS	b1100	Caused by an L2 miss when an OZQ miss already existed
SAME_INDEX	b1101	Caused by an L2 miss when a miss to the same index already existed
FRC_RECIRC	b1110	Caused by an L2 miss when a force recirculate already existed
—	b1111	(* nothing will be counted *)

L2_GOT_RECIRC_IFETCH

- **Title:** Instruction Fetch Recirculates Received by L2
- **Category:** L2 Unified Cache/L2 Cache Set 4 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xba, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of instruction fetch recirculates received by L2.
- **NOTE:** This is a restricted set 4 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4. Shares Event Code with L2_STORE_HIT_SHARED.

Table 11-81. Unit Masks for L2_GOT_RECIRC_IFETCH

Extension	PMC.umask [19:16]	Description
ANY	b1xxx	Instruction fetch recirculates received by L2

L2_GOT_RECIRC_OZQ_ACC

- **Title:** Counts OZQ Accesses Recirculated to L1D
- **Category:** L2 Unified Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb6, **Max. Inc/Cyc:** 1
- **Definition:** Counts number of OZQ accesses successfully recirculated to L1D.
- **NOTE:** This is a restricted set 2 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4.

L2_IFET_CANCELS

- **Title:** Instruction Fetch Cancels by the L2.
- **Category:** L2 Unified Cache/L2 Cache Set 0 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xa1,0xa5,0xa9,0xad, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of total instruction fetch cancels by L2
- **NOTE:** This is a restricted set 0 L2 Cache event. In order to measure this event, one of the L2_OZQ_CANCEL events or this event must be measured by PMD4.

Table 11-82. Unit Masks for L2_IFET_CANCELS

Extension	PMC.umask [19:16]	Description
ANY	b000x	Total instruction fetch cancels by L2
BYPASS	b001x	ifetch cancels due to bypassing
DIDNT_RECIR	b0100	ifetch cancels because it did not recirculate
RECIR_OVER_SUB	b0101	ifetch cancels because of recirculate oversubscription
ST_FILL_WB	b0110	ifetch cancels due to a store or fill or write back
DATA_RD	b0111	ifetch/prefetch cancels due to a data read
PREEMPT	b10xx	ifetch cancels due to preempts
CHG_PRIO	b1100	ifetch cancels due to change priority
IFETCH_BYP	b1101	Due to ifetch bypass during last clock
—	b1110-b1111	(* nothing will be counted *)

L2_INST_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
- **Definition:** Counts the number of instruction requests to L2 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, L2_INST_DEMAND_READS and L1I_READS line up in time. If a demand fetch does not have an L2ITLB miss, L2_INST_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 L2_INST_DEMAND_READS and L1I_READS is not deterministic.

L2_INST_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
- **Definition:** Provides information about the number of prefetch requests issued to the unified L2 cache. The reported number includes streaming and non-streaming prefetches.
- **NOTE:** This event is qualified with IBRP1.

L2_ISSUED_RECIRC_IFETCH

- **Title:** Instruction Fetch Recirculates Issued by L2
- **Category:** L2 Unified Cache/L2 Cache Set 4 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb9, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of instruction fetch recirculates issued by L2.
- **NOTE:** This is a restricted set 4 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4. Shares Event Code with L2_BAD_LINES_SELECTED.

Table 11-83. Unit Masks for L2_ISSUED_RECIRC_IFETCH

Extension	PMC.umask [19:16]	Description
ANY	b1xxx	Instruction fetch recirculates issued by L2

L2_ISSUED_RECIRC_OZQ_ACC

- **Title:** Count Number of Times a Recirculate Issue Was Attempted and Not Preempted
- **Category:** L2 Unified Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb5, **Max. Inc/Cyc:** 1
- **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 L2_OZQ_CANCELS*.DIDNT_RECIRC for the total number of times the L2 issue logic attempted to issue a recirculate.
- **NOTE:** This is a restricted set 2 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4.

L2_L3ACCESS_CANCEL

- **Title:** Canceled L3 Accesses
- **Category:** L2 Unified Cache/L2 Cache Set 1 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb0, **Max. Inc/Cyc:** 1
- **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.
- **NOTE:** This is a restricted set 1 L2 Cache event. This event must be measured by PMD4.

Table 11-84. Unit Masks for L2_L3ACCESS_CANCEL

Extension	PMC.umask [19:16]	Description
—	b0000	(* nothing will be counted *)
SPEC_L3_BYP	b0001	Speculative L3 bypasses
FILLD_FULL	b0010	Filld being full
—	b0011	(* count is undefined *)
—	b0100	(* nothing will be counted *)
UC_BLOCKED	b0101	Uncacheable blocked L3 Accesses
INV_L3_BYP	b0110	Invalid L3 bypasses
EBL_REJECT	b1000	ebl rejects
ANY	b1001	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 L2 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.

Table 11-84. Unit Masks for L2_L3ACCESS_CANCEL (Continued)

Extension	PMC.umask [19:16]	Description
DFETCH	b1010	Data fetches
IFETCH	b1011	Instruction fetches
—	b1100-b1111	(* nothing will be counted *)

L2_MISSES

- **Title:** L2 Misses
- **Category:** L2 Unified Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xcb, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of L2 cache misses (in terms of the number of L2 cache line requests sent to L3). It includes misses caused by instruction fetch/prefect and data read/write operations. It does not include L1 misses to uncacheable or write-coalescing addresses.

L2_OPS_ISSUED

- **Title:** Operations Issued By L2
- **Category:** L2 Unified Cache/L2 Cache Set 4 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb8, **Max. Inc/Cyc:** 4
- **Definition:** Counts the number of operations issued by L2 as specified by the operation type (i.e operations which were valid in the L2 pipe stage. So even if they are canceled later on, they will be counted. Fires only for operations which hit in L2; i.e. OzQ is handling them).
- **NOTE:** This is a restricted set 4 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4. Shares Event Code with L2_BYPASS.

Table 11-85. Unit Masks for L2_OPS_ISSUED

Extension	PMC.umask [19:16]	Description
INT_LOAD	b1000	Count only valid integer loads
FP_LOAD	b1001	Count only valid floating-point loads
RMW	b1010	Count only valid read_modify_write stores
STORE	b1011	Count only valid non-read_modify_write stores
NST_NLD	b1100	Count only valid non-load, no-store accesses
—	b1101-b1111	(* nothing will be counted *)

L2_OZDB_FULL

- **Title:** L2 OZ Data Buffer Is Full
- **Category:** L2 Unified Cache/L2 Cache Set 5 **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xbd, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of times L2 Oz Data Buffer is full.
- **NOTE:** This is a restricted set 5 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4.

Table 11-86. Unit Masks for L2_OZDB_FULL

Extension	PMC.umask [19:16]	Description
THIS	b0000	L2 OZ Data Buffer is full
—	b0001-b1111	(* count is undefined *)

L2_OZQ_ACQUIRE

- **Title:** Clocks With Acquire Ordering Attribute Existed in L2 OZQ
- **Category:** L2 Unified Cache/L2 Cache Set 0 **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xa2,0xa6,0xaa,0xae, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of clocks entries with “acquire” ordering attribute existed in the L2 OZ Queue.
- **NOTE:** This is a restricted set 0 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4.

L2_OZQ_CANCELS0

- **Title:** L2 OZQ Cancels (Late or Any)
- **Category:** L2 Unified Cache/L2 Cache Set 0 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xa0, **Max. Inc/Cyc:** 4
- **Definition:** Counts the number of total L2 OZ Queue Cancels (regardless of reason) or L2 OZ Queue Cancels due to a specific reason (based on umask).
- **NOTE:** This is a restricted set 0 L2 Cache event. Only 1 of the 3 L1_OZQ_CANCEL events may be measured at any given time. In order to measure this event, either L2_IFET_CANCELS or this event must be measured by PMD4.

Table 11-87. Unit Masks for L2_OZQ_CANCELS0

Extension	PMC.umask [19:16]	Description
ANY	bx000	Counts the total OZ Queue cancels
LATE_SPEC_BYP	bx001	Counts the late cancels caused by speculative bypasses
LATE_RELEASE	bx010	Counts the late cancels caused by releases
LATE_ACQUIRE	bx011	Counts the late cancels caused by acquires
LATE_BYP_EFFRELEASE	bx100	Counts the late cancels caused by L1D to L2A bypass effective releases
—	bx101-bx111	(* nothing will be counted *)

L2_OZQ_CANCELS1

- **Title:** L2 OZQ Cancels (Specific Reason Set 1)
- **Category:** L2 Unified Cache/L2 Cache Set 0 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xac, **Max. Inc/Cyc:** 4
- **Definition:** Counts the number of total L2 OZ Queue Cancels due to a specific reason (based on umask).
- **NOTE:** This is a restricted set 0 L2 Cache event. Only 1 of the 3 L1_OZQ_CANCEL events may be measured at any given time. In order to measure this event, either L2_IFET_CANCELS or this event must be measured by PMD4.

Table 11-88. Unit Masks for L2_OZQ_CANCELS1

Extension	PMC.umask [19:16]	Description
REL	b0000	Caused by release
BANK_CONF	b0001	Bank conflicts
L2D_ST_MAT	b0010	A store match in L2D
—	b0011	(* nothing will be counted *)
SYNC	b0100	Caused by sync.i
HPW_IFETCH_CONF	b0101	A ifetch conflict (canceling HPW?)
CANC_L2M_ST	b0110	Caused by a canceled store in L2M
L1_FILL_CONF	b0111	An L1 fill conflict
ST_FILL_CONF	b1000	A store fill conflict
CCV	b1001	A ccv
SEM	b1010	A semaphore
L2M_ST_MAT	b1011	A store match in L2M
MFA	b1100	A memory fence instruction
L2A_ST_MAT	b1101	A store match in L2A
L1DF_L2M	b1110	L1D fill in L2M
ECC	b1111	ECC hardware detecting a problem

L2_OZQ_CANCELS2

- **Title:** L2 OZQ Cancels (Specific Reason Set 2)
- **Category:** L2 Unified Cache/L2 Cache Set 0 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xa8, **Max. Inc/Cyc:** 4
- **Definition:** Counts the number of total L2 OZ Queue due to a specific reason (based on umask).
- **NOTE:** This is a restricted set 0 L2 Cache event. Only 1 of the 3 L1_OZQ_CANCEL events may be measured at any given time. In order to measure this event, either L2_IFET_CANCELS or this event must be measured by PMD4.

Table 11-89. Unit Masks for L2_OZQ_CANCELS2

Extension	PMC.umask [19:16]	Description
RECIRC_OVER_SUB	b0000	Caused by a recirculate oversubscription
CANC_L2C_ST	b0001	Caused by a canceled store in L2C
L2C_ST_MAT	b0010	A store match in L2C
SCRUB	b0011	32/64 byte HPW/L2D fill which needs scrub
ACQ	b0100	Caused by an acquire
READ_WB_CONF	b0101	A write back conflict (canceling read?)
OZ_DATA_CONF	b0110	An OZ data conflict
—	b0111	(* nothing will be counted *)
L2FILL_ST_CONF	b1000	An L2fill and store conflict in L2C
DIDNT_RECIRC	b1001	Caused because it did not recirculate
WEIRD	b1010	Counts the cancels caused by attempted 5-cycle bypasses for non-aligned accesses and bypasses blocking recirculates for too long
—	b1011	(* nothing will be counted *)
OVER_SUB	b1100	Oversubscription
CANC_L2D_ST	b1101	Caused by a canceled store in L2D
—	b1110	(* nothing will be counted *)
D_IFET	b1111	A demand ifetch

L2_OZQ_FULL

- **Title:** L2 OZQ Is Full
- **Category:** L2 Unified Cache/L2 Cache Set 5 **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xbc, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of times L2D Oz Queue is full.
- **NOTE:** This is a restricted set 5 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4.

Table 11-90. Unit Masks for L2_OZQ_FULL

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

L2_OZQ_RELEASE

- **Title:** Clocks With Release Ordering Attribute Existed in L2 OZQ
- **Category:** L2 Unified Cache/L2 Cache Set 0**IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xa3,0xa7,0xab,0xaf**Max. Inc/Cyc:**1
- **Definition:** Counts the number of clocks entries with “release” ordering attribute existed in the L2 OZ Queue.

- **NOTE:** This is a restricted set 0 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4.

L2_REFERENCES

- **Title:** Requests Made To L2
- **Category:** L2 Unified Cache/L2 Cache Set 1 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb1, **Max. Inc/Cyc:** 4
- **Definition:** Counts the number of requests (data reads, data writes, instruction fetches and instruction prefetches) made from L2.
- **NOTE:** This is a restricted set 1 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4. Prefetches which are promoted to fetches are only counted once. Instruction fetches to the second half of a line will not be counted if the fetch for the first half is already counted. All secondary misses are counted for data references. A semaphore operation will be counted only once here. Only requests which are entered into the OZQ are counted here; i.e. recirculated operations will not be recounted. Uncacheable/WC accesses will not be counted. FROM_CCV, SETF, CCV, PTC_G,PTC_GA, FWB, MF, MFA, SYNCI, SYNCIA, PTCM, FC, CC operations are excluded.

L2_STORE_HIT_SHARED

- **Title:** Store Hit a Shared Line
- **Category:** L2 Unified Cache/L2 Cache Set 3 **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xba, **Max. Inc/Cyc:** 2
- **Definition:** Counts the number of times a store hit a shared line.
- **NOTE:** This is a restricted set 3 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4. Shares Event Code with L2_GOT_RECIRC_IFETCH.

Table 11-91. Unit Masks for L2_STORE_HIT_SHARED

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

L2_SYNTH_PROBE

- **Title:** Synthesized Probe
- **Category:** L2 Unified Cache**IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xb7**Max. Inc/Cyc:**1
- **Definition:** Counts the number of synthesized probes. A synthesized probe indicates that L2 received a fill from the bus cluster with a MESI state of I or P indicating that the fill was hit by an in-flight snoop. As such, L2 needs to “synthesize” a probe response back to the bus cluster once the line has been “used once”. For forward progress, L2 won't send the response until it has used the line once.
- **NOTE:** This is a restricted set 2 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4.

L2_VICTIMB_FULL

- **Title:** L2D Victim Buffer Is Full
- **Category:** L2 Unified Cache/L2 Cache Set 5 **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0xbe, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of times L2D Victim Buffer is full.
- **NOTE:** This is a restricted set 5 L2 Cache event. In order to measure this event, one of the events in this set must be measured by PMD4.

Table 11-92. Unit Masks for L2_VICTIMB_FULL

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

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
- **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.

L3_MISSES

- **Title:** L3 Misses
- **Category:** L3 Unified Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xdc, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of L3 cache misses. Includes misses caused by instruction fetch, data read/write, L2 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
- **Definition:** Counts the number of L3 cache read accesses.

Table 11-93. 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

Table 11-93. Unit Masks for L3_READS (Continued)

Extension	PMC.umask [19:16]	Description
—	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
- **Definition:** Counts the number of L3 accesses. Includes instruction fetch/prefetch, data read/write and L2 write backs.

L3_WRITES

- **Title:** L3 Writes
- **Category:** L3 Unified Cache **IAR/DAR/OPC:** Y/Y/Y
- **Event Code:** 0xde, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of L3 cache write accesses.

Table 11-94. 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 L2 write backs, includes L3 read for ownership requests that satisfy stores)
DATA_WRITE.MISS	b0110	L3 Store Misses (excludes L2 write backs, includes L3 read for ownership requests that satisfy stores)
DATA_WRITE.ALL	b0111	L3 Store References (excludes L2 write backs, includes L3 read for ownership requests that satisfy stores)
—	b1000	(* nothing will be counted *)

Table 11-94. Unit Masks for L3_WRITES (Continued)

Extension	PMC.umask [19:16]	Description
L2_WB.HIT	b1001	L2 Write Back Hits
L2_WB.MISS	b1010	L2 Write Back Misses
L2_WB.ALL	b1011	L2 Write 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
- **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).
- **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.

MEM_READ_CURRENT

- **Title:** Current Mem Read Transactions On Bus
- **Category:** Frontside Bus **IAR/DAR/OPC:** N/N/N
- **Event Code:** 0x89, **Max. Inc/Cyc:** 1
- **Definition:** Counts the number of current memory read transactions (BRC) on the bus.

Table 11-95. Unit Masks for MEM_READ_CURRENT

Extension	PMC.umask [19:16]	Description
—	bxx00	(* nothing will be counted *)
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
- **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
- **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
- **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
- **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
- **Definition:** Counts the number of current RSE registers before an RSE_EVENT_RETIRED occurred. The Itanium 2 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

- **Definition:** Counts the number of current RSE registers before an RSE_EVENT_RETIRED occurred. The Itanium 2 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
- **Definition:** Counts the number of current RSE registers before an RSE_EVENT_RETIRED occurred. The Itanium 2 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
- **Definition:** Counts the number of dirty RSE registers before an RSE_EVENT_RETIRED occurred. The Itanium 2 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
- **Definition:** Counts the number of dirty RSE registers before an RSE_EVENT_RETIRED occurred. The Itanium 2 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
- **Definition:** Counts the number of dirty RSE registers before an RSE_EVENT_RETIRED occurred. The Itanium 2 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
- **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
- **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 11-96. 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
- **Definition:** Counts the number of srlz.i instructions (because it causes a microtrap and an xpnflush fires).

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
- **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
- **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 11-97. 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).
MLI	b1xxx	Count syllables not dispersed due to MLI bundle and resteers to non-0 syllable. Dispersal takes a 1 syllable hit for each MLI 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
- **Definition:** Counts the number of syllables which were overcounted in explicit and/or implicit stop bits portion of SYLL_NOT_DISPERSED.

Table 11-98. 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

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
- **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
- **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.

Model-Specific and Optional Features 12

This chapter describes model-specific features for the Itanium 2 processor.

12.1 Memory Attributes

Uncacheable Exported (UCE) is an optional feature of the Itanium architecture. The Itanium 2 processor supports WB, UC, and WC memory attributes. The UCE memory attribute is also supported, except with semaphore operations. Semaphore operations to a UCE page will fault. Otherwise, a UCE memory attribute can be used, but will behave as a UC attribute.

For more information regarding UCE behavior, please refer to Section 4.4, “Memory Attributes” of the *Intel® Itanium® Architecture Software Developer’s Manual, Volume 2: System Architecture.*

12.2 Purge Behavior of ptc.e

Purge behavior is model-specific. The Itanium 2 processor supports the following page sizes for purges or inserts:

- 4K, 8K, 16K, 64K, 256K, 1M, 4M, 16M, 64M, 256M, 1G, and 4G.

A `ptc.e` will cause all translation caches of all data and instruction TLB levels to be flushed in a single iteration.

12.3 Data Debug Break

The architecture gives freedom with the data debug break behavior. The Itanium 2 processor takes a data debug break fault on any memory access that crosses a 16 byte boundary when data breakpoints are enabled. This break occurs without regard to any addresses currently in the data debug break registers.

12.4 CPUID Values

On the Itanium 2 processor, the CPUID register contains the following processor identification information:

- CPUID registers 0 and 1 specify a vendor name, in ASCII, for the processor implementation.
- CPUID register 2 is an ignored register (reads from this register return zero).
- CPUID register 3 contains several fields indicating version information related to the processor implementation. Table 12-1 and Table 12-2 specify the definitions of each field. For revision information, please see the Itanium 2 processor specification.

- CPUID register 4 provides general application-level information about processor features. As shown in Table 12-3, it is a set of flag bits used to indicate if a given feature is supported in the processor model.

Table 12-1. Itanium® 2 Processor CPUID Register 3 Values

Bit Field	Value	Description
7:0	0x04	Number of supported CPUID registers
15:8	—	Processor revision
23:16	—	Processor model
31:24	—	Processor family
39:32	0x00	Architectural revision
63:40	0x00	Reserved

Table 12-2. Itanium® 2 Processor Family and Model Values

Family	Model	Description
0x07	0x00	Itanium® Processor
0x1f	0x00	Itanium 2 Processor (1.5M) or Itanium 2 Processor (1 GHz, 3M)
0x1f	0x01	Itanium 2 Processor (1.30 GHz, 3M) or Itanium 2 Processor (4 M) or Itanium 2 Processor (6 M) or Low Voltage Itanium 2 Processor

Table 12-3. Itanium® 2 Processor CPUID Register 4 Values

Field	Bits	Description
lb	0	Indicates brl instruction is implemented; OS does not need to emulate it.
sd	1	Processor implements spontaneous deferral (see Section 5.5.5, “Deferral of Speculative Load Faults” of the *Intel® Itanium® Architecture Software Developer’s Manual, Volume 2: System Architecture*.
ao	2	Processor implements 16-byte atomic operations (see “ld–Load”, “st–Store”, and “cmpxchg–Compare and Exchange” instructions in *Volume 3: Instruction Set Reference* of the *Intel® Itanium® Architecture Software Developer’s Manual*.
rv	63:3	Reserved.

Table 12-4 provides information on how to decode return values of the IA-32 CPUID instruction for the Itanium 2 processor’s caches.

Table 12-4. Encoding of IA-32 CPUID Cache Return Values

Return Value	Cache Description
0x67	L1D: 16KB 4-way, 64B line size
0x77	L1I: 16KB 4-way, 64B line size
0x7e	L2: 256KB 8-way, 128B line size
0x8d	L3: 3MB 12-way, 128B line size

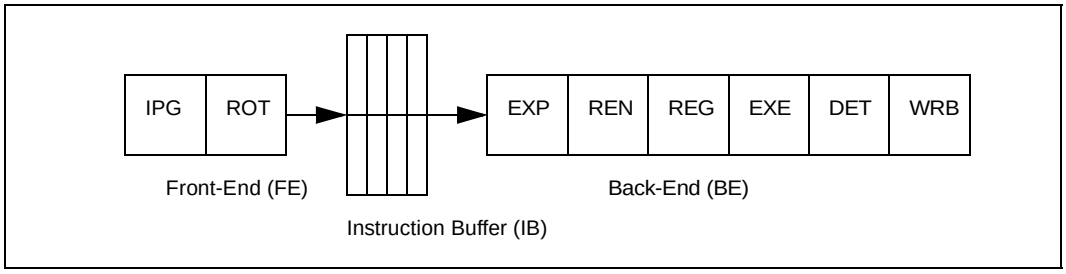
Itanium® 2 Processor Pipeline A

The core pipeline is eight stages deep, with some other micropipelines working asynchronously to the core pipeline.

A.1 Core Pipeline

The core pipeline is separated into a front-end (FE) and a back-end (BE). The FE and BE are separated by an instruction buffer (IB).

Figure A-1. Core Pipeline of the Itanium® 2 Processor



The core pipeline consists of eight stages:

IPG: Instruction pointer generation

ROT: instruction rotation

EXP: Instruction template decode, expand, and disperse

REN: Rename (for register stack and rotating registers) and decode

REG: Register file read

EXE: ALU execution

DET: Last stage for exception detection

WRB: Write back

A.2 Pipeline Stages

A.2.1 IPG STAGE

The Instruction Pointer Generation (*IPG*) stage delivers an instruction pointer to the L1I. The value of the instruction pointer may come from one of several places: corrected target or fall through address (to correct branch misprediction), the address of exception handler in case of exceptions, static and dynamically predicted addresses, or the next sequential address. During this stage, the L1I, ISB, and L1 ITLB are accessed.

The L1I to *IPG* interface always aligns the bundle-pair on even bundle (i.e. 32 byte) address boundaries. A branch that targets a bundle on an odd boundary bundle will fetch the bundle-pair from the lower even-bundle address. As a consequence, only 1 useful bundle (instead of the maximum of two) will be delivered to *IPG* at such a branch target.

A.2.2 ROT STAGE

The Rotate (*ROT*) stage reads out the four ways of the instruction cache array and the ISB data and selects the correct way. Fetched instructions are rotated in order to align them for use in the *EXP* stage. This is necessary since the instruction disperse and decode unit in the *EXP* stage will look at two bundles to decide whether to issue zero, one, or two bundles from the rotation buffer. Therefore, there is a need to properly align the instructions in the rotation buffer since the decode unit always assumes that bundle 0 contains the leading instruction.

A.2.3 EXP STAGE

The Expand (*EXP*) stage decodes instruction templates and disperses up to 6 instructions to functional units. Due to resource constraints (such constraints are discussed in the next section), some fetched bundles may not get fully dispatched. These fetched but not dispatched bundles are "pushed back" into the *IB*. The number of bundles consumed by the *EXP* stage is fed back to the rotate buffer to determine which bundles to be presented to *EXP* stage the next cycle.

A.2.4 REN STAGE

The Rename (*REN*) stage translates virtual registers into physical registers by adding their values to the stack frame base and the rotating register base. Instruction decoding also occurs at this stage.

A.2.5 REG Stage

The Register Read (*REG*) stage delivers operands to all execution units. The data read is fed into a set of bypass muxes. There are two levels of bypass muxes. Data read from the register files and results from the *DET* and *WRB* stage are sent to the early bypass. Data generated from the current *EXE* stage and load data is fed into the late bypass muxes. Register and dynamic resource hazards are detected and issue is stalled if necessary. RSE loads and stores are injected in the pipeline in the *REG* stage.

A.2.6 EXE Stage

The Execution (*EXE*) stage is the ALU execution stage. Single cycle latency operations feed results to the late bypass muxes by the end of this stage, for use by subsequent integer ALU operations.

A.2.7 DET Stage

The Detection (*DET*) stage is the last stage where exception detection can occur. By the end of the *DET* stage, all potential exceptions are known in time to kill the write back of architectural state. Branch validation for incorrect branch direction is also handled at this stage. If a branch is mispredicted, (either because the prediction of predicate is wrong or the predicted target address is wrong), the actual resteer of the next IP address occurs at this stage. So a branch misprediction can cause six cycles of pipeline bubbles.

A.2.8 WRB Stage

The Write Back (*WRB*) stage writes results back to register files. Once an instruction completes the *WRB* stage, it is guaranteed to update the architectural processor-state.

A.3 Instruction Buffer (IB)

The main pipeline is *decoupled* between the *ROT* and *EXP* pipeline stages. The first two stages belong to the *Front-End (FE)*, and the remaining six stages are the *Back-End (BE)*. The *Instruction* Buffer (*IB*), a decoupling queue, links together the *FE* and the *BE*. This *IB* can hold 4 bundle-pairs (24 instructions). With the *IB*, the *FE* and the *BE* can work independently. Hence, instruction cache miss delays and taken branch penalties may be hidden by execution stalls incurred in the *BE*.

A.4 Micro-Pipelines

A.4.1 FPU Micro-Pipeline

The FPU pipeline is four stages deep (*FP1* to *FP4*), with write back performed in the fifth stage (*FWB*). The FPU is fully pipelined. In the *FP1* stage, an early examination of the numeric operands is performed to determine if the instruction can be numeric exception free.

Table A-1. FPU Pipeline

Core Pipeline	REG	EXE	DET	WRB		
FPU Pipeline		FP1	FP2	FP3	FP4	FWB

A.4.2 L1D Micro-Pipeline

In the *L1M* stage, the L1 data, tag and the L1 DTLB are accessed in parallel and deliver data to the execution units.

Table A-2. L1D Micro-Pipeline

Core Pipeline	REN	REG	EXE	DET	WRB
L1D Pipeline		L1I	L1M	L1D	WRB

A.4.3 L2 Micro-Pipeline

The first stage is used for L2 TLB accesses. The L2A stage arbitrates for the data array accesses. Demand fetches for instructions have the highest priority, followed by loads and prefetches. Data array access occurs in the L2M stage. The L2D stage is for way selection, and data delivery. The L2C stage is used for correction of ECC errors and for error detection.

Table A-3. L2 Micro-Pipeline

Core Pipeline	REG	EXE	DET	WRB			
L2 Pipeline		L2L	L2A	L2M	L2D	L2C	L2W