The RDMSR and WRMSR instructions can read and write the time-stamp counter, respectively, as an MSR (at MSR address 10H). In the Pentium 4, Intel Xeon, and P6 family processors, all 64-bits of the time-stamp counter can be read with the RDMSR instruction (just as with the RDTSC instruction). However, when the WRMSR instruction is used to write to the time-stamp counter, only the low order 32-bits of the time-stamp counter can be written to, and the high-order 32 bits cleared to all 0s.

15.8. PERFORMANCE MONITORING OVERVIEW

Performance monitoring was introduced to the IA-32 architecture in the Pentium processor with a set of model-specific performance-monitoring counter MSRs. These counters permit a selection of processor performance parameters to be monitored and measured. The information obtained from these counters can then be used for tuning system and compiler performance.

In the Intel P6 family of processors, the performance monitoring mechanism was modified and enhanced to permit a wider selection of events to be monitored and to allow greater control over the choice of the events to be monitored.

The Pentium 4 and Intel Xeon processors introduced a new performance monitoring mechanism and new set of performance events that can be counted.

The performance monitoring mechanisms and performance events defined for the Pentium, P6 family, Pentium 4, and Intel Xeon processors are not architectural. They are all model specific and are not compatible among the three IA-32 processor families.

The following sections describe the performance monitoring mechanisms for the Pentium 4, Intel Xeon, P6 family, and Pentium processors, respectively:

- Section 15.9., "Performance Monitoring (Pentium 4 and Intel Xeon Processors)"
- Section 15.11., "performance Monitoring (P6 Family Processor)"
- Section 15.12., "Performance Monitoring (Pentium Processors)"

15.9. PERFORMANCE MONITORING (PENTIUM 4 AND INTEL XEON PROCESSORS)

The performance monitoring mechanism provided in the Pentium 4 and Intel Xeon processors is considerably different from that provided in the P6 family and Pentium processors. While the general concept of selecting, filtering, counting, and reading performance events through the WRMSR, RDMSR, and RDPMC instructions is unchanged, the setup mechanism and MSR layouts are different and incompatible with the P6 family and Pentium processor mechanisms. Also, the RDPMC instruction has been enhanced to read the additional performance counters provided in the Pentium 4 and Intel Xeon processors and to allow faster reading of the counters.

The event monitoring mechanism provided with the Pentium 4 and Intel Xeon processors consists of the following facilities:

- The IA32_MISC_ENABLE MSR, which indicates the availability in an IA-32 processor of the performance monitoring and precise event-based sampling (PEBS) facilities.
- 45 event selection control (ESCR) MSRs for selecting events to be monitored with specific performance counters.
- 18 performance counter MSRs for counting events.
- 18 counter configuration control (CCCR) MSRs, with one CCCR associated with each performance counter. Each CCCR sets up its associated performance counter for a specific method or style of counting.
- A debug store (DS) save area in memory for storing PEBS records.
- The IA32_DS_AREA MSR, which establishes the location of the DS save area.
- The debug store (DS) feature flag (bit 21) returned by the CPUID instruction, which indicates the availability in an IA-32 processor of the DS mechanism.
- The IA32_PEBS_ENABLE MSR, which enables the PEBS facilities and replay tagging used in at-retirement event counting.
- A set of predefined events and event metrics that simplify the setting up of the performance counters to count specific events.

Table 15-4 lists the performance counters and their associated CCCRs, along with the ESCRs that select events to be counted for each performance counter. The predefined event metrics and events are listed in Table in Appendix A, *Performance-Monitoring Events*.

Table 15-4. Performance Counter MSRs and Associated CCCR and ESCR MSRs (Pentium 4 and Intel Xeon Processors)

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_BPU_COUNTER0	0	300H	MSR_BPU_CCCR0	360H	MSR_BSU_ESCR0 MSR_FSB_ESCR0 MSR_MOB_ESCR0 MSR_PMH_ESCR0 MSR_BPU_ESCR0 MSR_IS_ESCR0 MSR_ITLB_ESCR0 MSR_IX_ESCR0	7 6 2 4 0 1 3 5	3A0H 3A2H 3AAH 3ACH 3B2H 3B4H 3B6H 3C8H
MSR_BPU_COUNTER1	1	301H	MSR_BPU_CCCR1	361H	MSR_BSU_ESCR0 MSR_FSB_ESCR0 MSR_MOB_ESCR0 MSR_PMH_ESCR0 MSR_BPU_ESCR0 MSR_IS_ESCR0 MSR_ITLB_ESCR0 MSR_IX_ESCR0	7 6 2 4 0 1 3 5	3A0H 3A2H 3AAH 3ACH 3B2H 3B4H 3B6H 3C8H

Table 15-4. Performance Counter MSRs and Associated CCCR and ESCR MSRs (Pentium 4 and Intel Xeon Processors) (Contd.)

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_BPU_COUNTER2	2	302H	MSR_BPU_CCCR2	362H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_BPU_COUNTER3	3	303H	MSR_BPU_CCCR3	363H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_MS_COUNTER0	4	304H	MSR_MS_CCCR0	364H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER1	5	305H	MSR_MS_CCCR1	365H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER2	6	306H	MSR_MS_CCCR2	366H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_MS_COUNTER3	7	307H	MSR_MS_CCCR3	367H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_FLAME_ COUNTER0	8	308H	MSR_FLAME_CCCR0	368H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAAT_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
MSR_FLAME_ COUNTER1	9	309H	MSR_FLAME_CCCR1	369H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAAT_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
MSR_FLAME_COUNTE R2	10	30AH	MSR_FLAME_CCCR2	36AH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAAT_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H
MSR_FLAME_COUNTE R3	11	30BH	MSR_FLAME_CCCR3	36BH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAAT_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H
MSR_IQ_COUNTER0	12	30CH	MSR_IQ_CCCR0	36CH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0 MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH

Table 15-4. Performance Counter MSRs and Associated CCCR and ESCR MSRs (Pentium 4 and Intel Xeon Processors) (Contd.)

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_IQ_COUNTER1	13	30DH	MSR_IQ_CCCR1	36DH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0 MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER2	14	30EH	MSR_IQ_CCCR2	36EH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1 MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER3	15	30FH	MSR_IQ_CCCR3	36FH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1 MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER4	16	310H	MSR_IQ_CCCR4	370H	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0 MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER5	17	311H	MSR_IQ_CCCR5	371H	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1 MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH

The types of events that can be counted with these performance monitoring facilities are divided into two classes: non-retirement events and at-retirement events.

- Non-retirement events (see Table A-1) are events that occur any time during instruction execution (such as bus transactions or cache transactions).
- At-retirement events (see Table A-2) are events that are counted at the retirement stage of instruction execution, which allows finer granularity in counting events and capturing machine state. The at-retirement counting mechanism includes facilities for tagging µops that have encountered a particular performance event during instruction execution. Tagging allows events to be sorted between those that occurred on an execution path that resulted in architectural state being committed at retirement as well as events that occurred on an execution path where the results were eventually cancelled and never committed to architectural state (such as, the execution of a mispredicted branch).

The Pentium 4 and Intel Xeon processors' performance monitoring facilities support the three usage models described below. The first two models can be used to count both non-retirement and at-retirement events, the third model can be used only to count a subset of at-retirement events:

- **Event counting.** A performance counter is configured to count one or more types of events. While the counter is counting, software reads the counter at selected intervals to determine the number of events that have been counted between the intervals.
- **Non-precise event-based sampling.** A performance counter is configured to count one or more types of events and to generate an interrupt when it overflows. To trigger an overflow, the counter is preset to a modulus value that will cause the counter to overflow after a specific number of events have been counted. When the counter overflows, the processor generates a performance monitoring interrupt (PMI). The interrupt service routine for the PMI then records the return instruction pointer (RIP), resets the modulus, and restarts the counter. Code performance can be analyzed by examining the distribution of RIPs with a tool like the VTune™ Performance Analyzer.
- **Precise event-based sampling (PEBS).** This type of performance monitoring is similar to non-precise event-based sampling, except that a memory buffer is used to save a record of the architectural state of the processor whenever the counter overflows. The records of architectural state provide additional information for use in performance tuning. Precise event-based sampling can be used to count only a subset of at-retirement events.

The following sections describe the MSRs and data structures used for performance monitoring in the Pentium 4 and Intel Xeon processors, then describes how these facilities are used with the three usage models described above.

15.9.1. ESCR MSRs

The 45 ESCR MSRs (see Table 15-4) allow software to select specific events to be countered. Each ESCR is usually associated with a pair of performance counters (see Table 15-4), and each performance counter has several ESCRs associated with it (allowing the events to be counted to be selected from a variety of events).

Figure 15-7 shows the layout of an ESCR MSR. The functions of the flags and fields are as follows:

USR flag, bit 2

When set, events are counted when the processor is operating at a current privilege level (CPL) of 1, 2, or 3. These privilege levels are generally used by application code and unprotected operating system code.

OS flag, bit 3

When set, events are counted when the processor is operating at CPL of 0. This privilege level is generally reserved for protected operating system code. (When both the OS and USR flags are set, events are counted at all privilege levels.)

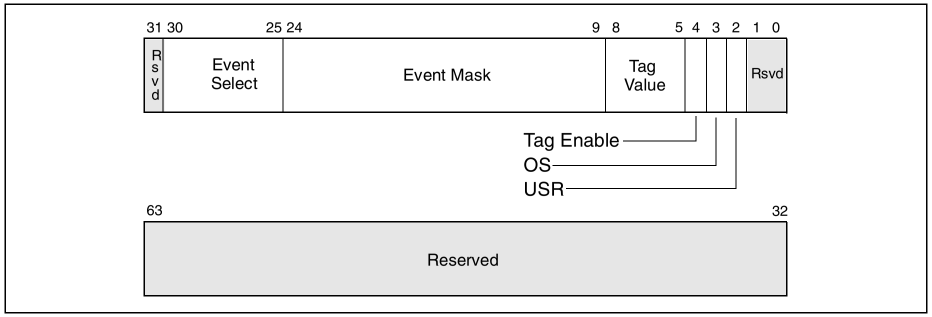

Figure 15-7. Event Selection Control Register (ESCR) (Pentium 4 and Intel Xeon processors)

Tag Enable, bit 4

When set, enables tagging of µops to assist in at-retirement event counting; when clear, disables tagging. See Section 15.9.7., "At-Retirement Counting".

Tag Value field, bits 5 through 8

Selects a tag value to associate with a µop to assist in at-retirement event counting.

Event Mask field, bits 9 through 24

Selects events to be counted from the event class selected with the event select field.

Event Select field, bits 25 through 30)

Selects a class of events to be counted. The events within this class that are counted are selected with the event mask field.

When setting up an ESCR, the event select field is used to select a specific class of events to count, such as retired branches. The event mask field is then used to select one or more of the specific events within the class to be counted. For example, when counting retired branches, four different events can be counted: branch not taken predicted, branch not taken mispredicted, branch taken predicted, and branch taken mispredicted. The OS and USR flags allow counts to be enabled for events that occur when operating system code and/or application code are being executed. If neither the OS nor USR flag is set, no events will be counted.

The ESCRs are initialized to all 0s on reset. The flags and fields of an ESCR are configured by writing to the ESCR using the WRMSR instruction. Table 15-4 gives the addresses of the ESCR MSRs.

NOTE

Writing to an ESCR MSR does not enable counting with its associated performance counter; it only selects the event or events to be counted. The CCCR for the selected performance counter must also be configured. Configuration of the CCCR includes selecting the ESCR and enabling the counter.

15.9.2. Performance Counters

The performance counters in conjunction with the counter configuration control registers (CCCRs) are used for filtering and counting the events selected by the ESCRs. The Pentium 4 and Intel Xeon processors provide 18 performance counters organized into 9 pairs. A pair of performance counters is associated with a particular subset of events and ESCR's (see Table 15-4). The counter pairs are partitioned into four groups:

- The BPU group, includes two performance counter pairs:
  - — MSR_BPU_COUNTER0 and MSR_BPU_COUNTER1
  - — MSR_BPU_COUNTER2 and MSR_BPU_COUNTER3.
- The MS group, includes two performance counter pairs:
  - — MSR_MS_COUNTER0 and MSR_MS_COUNTER1.
  - — MSR_MS_COUNTER2 and MSR_MS_COUNTER3.
- The FLAME group, includes two performance counter pairs:
  - — MSR_FLAME_COUNTER0 and MSR_FLAME_COUNTER1.
  - — MSR_FLAME_COUNTER2 and MSR_FLAME_COUNTER3.
- The IQ group, includes three performance counter pairs:
  - — MSR_IQ_COUNTER0 and MSR_IQ_COUNTER1.
  - — MSR_IQ_COUNTER2 and MSR_IQ_COUNTER3.
  - — MSR_IQ_COUNTER4 and MSR_IQ_COUNTER5.

The MSR_IQ_COUNTER4 counter in the IQ group provides support for the PEBS.

Alternate counters in each group can be cascaded: the first counter in one pair can start the first counter in the second pair and vice versa. A similar cascading is possible for the second counters in each pair. For example, within the BPU group of counters, MSR_BPU_COUNTER0 can start MSR_BPU_COUNTER2 and vice versa, and MSR_BPU_COUNTER1 can start MSR_BPU_COUNTER3 and vice versa (see Section 15.9.6.6., "Cascading Counters"). The cascade flag in the CCCR register for the performance counter enables the cascading of counters.

Each performance counter is 40-bits wide (see Figure 15-8). The RDPMC instruction has been enhanced in the Pentium 4 and Intel Xeon processors to allow reading of either the full counter-width (40-bits) or the low 32-bits of the counter. Reading the low 32-bits is faster than reading the full counter width and is appropriate in situations where the count is small enough to be contained in 32 bits.

The RDPMC instruction can be used by programs or procedures running at any privilege level and in virtual-8086 mode to read these counters. The PCE flag in control register CR4 (bit 8) allows the use of this instruction to be restricted to only programs and procedures running at privilege level 0.

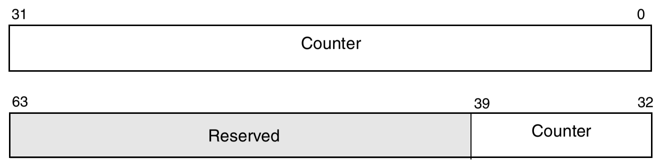

Figure 15-8. Performance Counter (Pentium 4 and Intel Xeon Processors)

The RDPMC instruction is not serializing or ordered with other instructions. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDPMC instruction operation is performed.

Only the operating system, executing at privilege level 0, can directly manipulate the performance counters, using the RDMSR and WRMSR instructions. A secure operating system would clear the PCE flag during system initialization to disable direct user access to the performance-monitoring counters, but provide a user-accessible programming interface that emulates the RDPMC instruction.

Some uses of the performance counters require the counters to be preset before counting begins (that is, before the counter is enabled). This can be accomplished by writing to the counter using the WRMSR instruction. To set a counter to a specified number of counts before overflow, enter a 2s complement negative integer in the counter. The counter will then count from the preset value up to -1 and overflow. Writing to a performance counter in a Pentium 4 or Intel Xeon processor with the WRMSR instruction causes all 40 bits of the counter to be written.

15.9.3. CCCR MSRs

Each of the 18 performance counters in a Pentium 4 or Intel Xeon processor has one CCCR MSR associated with it (see Table 15-4). The CCCRs control the filtering and counting of events as well as interrupt generation. Figure 15-9 shows the layout of an CCCR MSR. The functions of the flags and fields are as follows:

Enable flag, bit 12
When set, enables counting; when clear, the counter is disabled. This flag is cleared on reset.

ESCR Select field, bits 13 through 15
Identifies the ESCR to be used to select events to be counted with the counter associated with the CCCR.

Compare flag, bit 18
When set, enables filtering of the event count; when clear, disables filtering. The filtering method is selected with the threshold, complement, and edge flags.

Complement flag, bit 19
Selects how the incoming event count is compared with the threshold value. When set, event counts that are less than or equal to the threshold value result in a single count being delivered to the performance counter; when clear, counts greater than the threshold value result in a count being delivered to the performance counter (see Section 15.9.6.2., “Filtering Events”). The complement flag is not active unless the compare flag is set.

Threshold field, bits 20 through 23
Selects the threshold value to be used for comparisons. The processor examines this field only when the compare flag is set, and uses the complement flag setting to determine the type of threshold comparison to be made. The useful range of values that can be entered in this field depend on the type of event being counted (see Section 15.9.6.2., “Filtering Events”).

Edge flag, bit 24
When set, enables rising edge (false-to-true) edge detection of the threshold comparison output for filtering event counts; when clear, rising edge detection is disabled. This flag is active only when the compare flag is set.

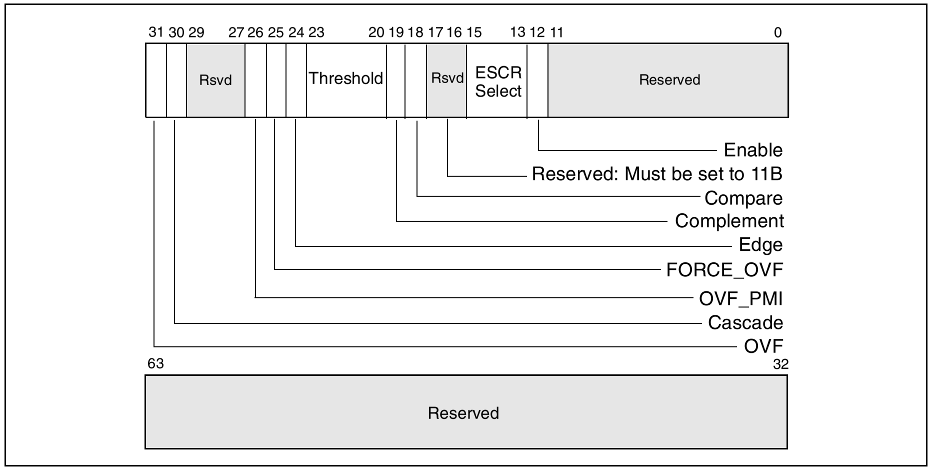

Figure 15-9. Counter Configuration Control Register (CCCR)

FORCE_OVF flag, bit 25
When set, forces a counter overflow on every counter increment; when clear, overflow only occurs when the counter actually overflows.

OVF_PMI flag, bit 26
When set, causes a performance monitor interrupt (PMI) to be generated when the counter overflows occurs; when clear, disables PMI generation. Note that the PMI is generated on the next event count after the counter has overflowed.

Cascade flag, bit 30
When set, enables counting on one counter of a counter pair when its alternate counter in the other the counter pair in the same counter group overflows (see Section 15.9.2., “Performance Counters” for further details); when clear, disables cascading of counters.

OVF flag, bit 31
Indicates that the counter has overflowed when set. This flag is a sticky flag that must be explicitly cleared by software.

The CCCRs are initialized to all 0s on reset.

The events that an enabled performance counter actually counts are selected and filtered by the following flags and fields in the ESCR and CCCR registers and in the qualification order given:

1. The event select and event mask fields in the ESCR select a class of events to be counted and one or more event types within the class, respectively.
2. The OS and USR flags in the ESCR selected the privilege levels at which events will be counted.

3. The ESCR select field of the CCCR selects the ESCR. Since each counter has several ESCRs associated with it, one ESCR must be chosen to select the classes of events that may be counted.
4. The compare and complement flags and the threshold field of the CCCR select an optional threshold to be used in qualifying an event count.
5. The edge flag in the CCCR allows events to be counted only on rising-edge transitions.

The qualification order in the above list implies that the filtered output of one "stage" forms the input for the next. For instance, events filtered using the privilege level flags can be further qualified by the compare and complement flags and the threshold field, and an event that matched the threshold criteria, can be further qualified by edge detection.

The uses of the flags and fields in the CCCRs are discussed in greater detail in Section 15.9.6., "Programming the Performance Counters for Non-Retirement Events".

15.9.4. Debug Store (DS) Mechanism

The debug store (DS) mechanism was introduced in the Pentium 4 and Intel Xeon processors to allow various types of information to be collected in memory-resident buffers for use in debugging and tuning programs. For the Pentium 4 and Intel Xeon processors, the DS mechanism is used to collect two types of information: branch records and precise event-based sampling (PEBS) records (see Section 15.5.7., "Branch Trace Store (BTS)" and Section 15.9.8., "Precise Event-Based Sampling (PEBS)" for a description of these facilities.) The availability of the DS mechanism in a processor is indicated with the DS feature flag (bit 21) returned by the CPUID instruction.

Records collected with the DS mechanism are saved in the DS save area (see Section 15.9.5., "DS Save Area").

15.9.5. DS Save Area

The debug store (DS) save area is a software-designated area of memory that is used to collect the following two types of information:

- **Branch Records.** When the BTS flag in the IA32_DEBUGCTL MSR is set, a branch record is stored in the BTS buffer in the DS save area whenever a taken branch, interrupt, or exception is detected.
- **PEBS Records.** When a performance counter is configured for PEBS, a PEBS record is stored in the PEBS buffer in the DS save area whenever a counter overflow occurs. This record contains the architectural state of the processor (state of the 8 general purpose registers, EIP register, and EFLAGS register) at the time of the event that caused the counter to overflow. When the state information has been logged, the counter is automatically reset to a preselected value, and event counting begins again. This feature is available only for a subset of the Pentium 4 and Intel Xeon processors' performance events.

NOTE

DS save area and recording mechanism is not available in the SMM. The feature is disabled on transition to the SMM mode. Similarly DS recording is disabled on the generation of a machine check exception and is cleared on processor RESET and INIT. DS recording is available in real address mode.

The BTS and PEBS facilities may not be available on all IA-32 processors. The availability of these facilities is indicated with the BTS_UNAVAILABLE and PEBS_UNAVAILABLE flags, respectively, in the IA32_MISC_ENABLE MSR (see Table B-1).

The DS save area is divided into three parts (see Figure 15-10): buffer management area, branch trace store (BTS) buffer, and PEBS buffer. The buffer management area is used to define the location and size of the BTS and PEBS buffers. The processor then uses the buffer management area to keep track of the branch and/or PEBS records in their respective buffers and to record the performance counter reset value. The linear address of the first byte of the DS buffer management area is specified with the IA32_DS_AREA MSR.

The fields in the buffer management area are as follows:

BTS buffer base
: Linear address of the first byte of the BTS buffer. This address should point to a natural doubleword boundary.

BTS Index
: Linear address of the first byte of the next BTS record to be written to. Initially, this address should be the same as the address in the BTS buffer base field.

BTS absolute maximum
: Linear address of the next byte past the end of the BTS buffer. This address should be a multiple of the BTS record size (12 bytes) plus 1.

BTS interrupt threshold
: Linear address of the BTS record on which an interrupt is to be generated. This address must point to an offset from the BTS buffer base that is a multiple of the BTS record size. Also, it must be several records short of the BTS absolute maximum address to allow a pending interrupt to be handled prior to processor writing the BTS absolute maximum record.

PEBS buffer base
: Linear address of the first byte of the PEBS buffer. This address should point to a natural doubleword boundary.

PEBS index
: Linear address of the first byte of the next PEBS record to be written to. Initially, this address should be the same as the address in the PEBS buffer base field.

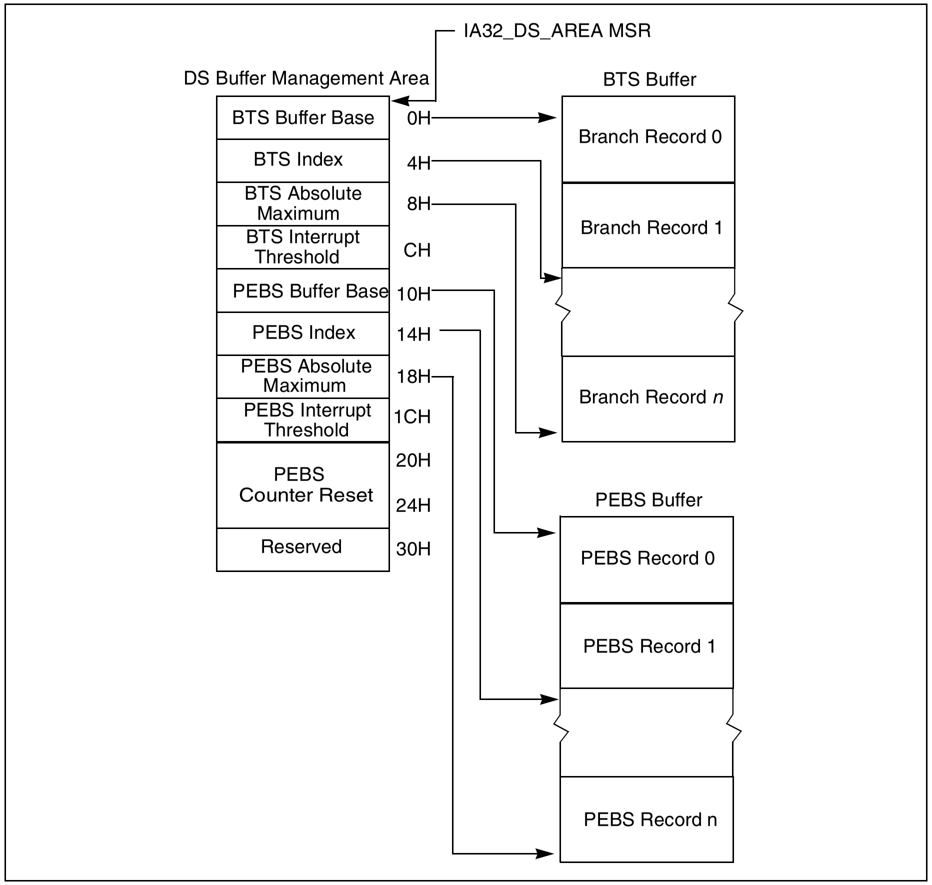

Figure 15-10. DS Save Area

PEBS absolute maximum
Linear address of the next byte past the end of the PEBS buffer. This address should be a multiple of the PEBS record size (40 bytes) plus 1.

PEBS interrupt threshold
Linear address of the PEBS record on which an interrupt is to be generated. This address must point to an offset from the PEBS buffer base that is a multiple of the PEBS record size. Also, it must be several records short of the PEBS absolute maximum address to allow a pending interrupt to be handled prior to processor writing the PEBS absolute maximum record.

PEBS counter reset value
A 40-bit value that the counter is to be reset to after state information has collected following counter overflow. This value allows state information to be collected after a preset number of events have been counted.

Figures 15-11 shows the structure of a 12-byte branch record in the BTS buffer. The fields in each record are as follows:

Last Branch From
Linear address of the instruction from which the branch, interrupt, or exception was taken.

Last Branch To Linear address of the branch target or the first instruction in the interrupt or exception service routine.

Branch Predicted
Bit 4 of field indicates whether the branch that was taken was predicted (set) or not predicted (clear).

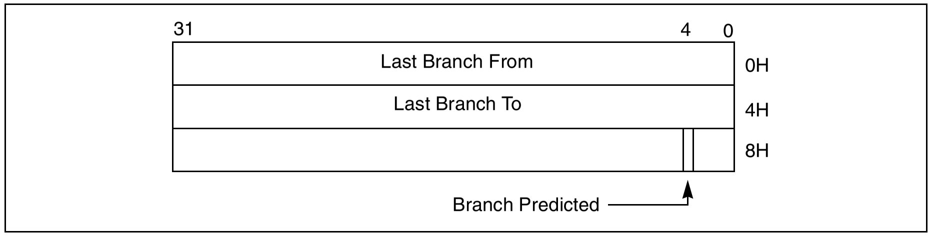

Figure 15-11. Branch Trace Record Format

Figures 15-12 shows the structure of the 40-byte PEBS records. Nominally the register values are those at the beginning of the instruction that caused the event. However, there are cases where the registers may be logged in a partially modified state. The linear IP field shows the value in the EIP register translated from an offset into the current code segment to a linear address.

15.9.6. Programming the Performance Counters for Non-Retirement Events

To program a performance counter and begin counting events, software must perform the following operations.

1. Select the event or events to be counted.
2. For each event, select an ESCR that supports the event using the values in the ESCR Restrictions row in Table A-1.

3. Match the CCCR Select value and ESCR name in Table A-1 to the values listed ESCR Name and ESCR No. columns in Table 15-4, to select a CCCR and performance counter.
4. Set up an ESCR for the specific event or events to be counted and the privilege levels they are to be counted at.
5. Set up the CCCR for the performance counter to be used to count the events, by selecting the chosen the ESCR and selecting the desired event filters.
6. Set up the CCCR for optional cascading of event counts, so that when the selected counter overflows its alternate counter starts counting.
7. Set up the CCCR to generate an optional performance monitor interrupt (PMI) when the counter overflows. (If PMI generation is enabled, the local APIC must be set up to deliver the interrupt to the processor and a handler for the interrupt must be in place.)
8. Enable the counter to begin counting.

31 — 0	
EFLAGS	0H
Linear IP	4H
EAX	8H
EBX	CH
ECX	10H
EDX	14H
ESI	18H
EDI	1CH
EBP	20H
ESP	24H

Figure 15-12. PEBS Record Format

15.9.6.1. SELECTING EVENTS TO COUNT

Table A-1 lists a set of non-retirement events for the Pentium 4 and Intel Xeon processors. For each event listed in Table A-1, specific setup information is provided. Figure 15-13 gives an example of one of the non-retirement events from Table A-1.

In Tables A-1 and A-2, the name of the event is listed in the Event Name column and various parameters that define the event and other information are listed in the Event Parameters column. The Parameter Value and Description columns give specific parameters for the event and additional description information. The entries in the Event Parameters column are described below.

DEBUGGING AND PERFORMANCE MONITORING

ESCR Restrictions

Lists the ESCRs that can be used to program the event. Typically only one ESCR is needed to count an event.

Event Name	Event Parameters	Parameter Value	Description
Branch_retired			Counts the retirement of a branch. Specify one or more mask bits to select any combination of branch taken, not-taken, predicted and mispredicted.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	See Table 15-3 for the addresses of the ESCR MSRs
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	The counter numbers associated with each ESCR are provided. The performance counters and corresponding CCCRs can be obtained from Table 15-3.
	ESCR Event Select	06H	ESCR[31:25]
	ESCR Event Mask	Bit 0: MMNP 1: MMNM 2: MMTP 3: MMTM	ESCR[24:9], Branch Not-taken Predicted, Branch Not-taken Mispredicted, Branch Taken Predicted, Branch Taken Mispredicted.
	CCCR Select	05H	CCCR[15:13]
	Event Specific Notes		P6: EMON_BR_INST_RETIRED
	Can Support PEBS	No	
	Requires Additional MSRs for Tagging	No	

Figure 15-13. Event Example

Counter numbers per ESCR

Lists which performance counters are associated with each ESCR. Table 15-4 gives the name of the counter and CCCR for each counter number. Typically only one counter is needed to count the event.

ESCR Event Select

Gives the value to be placed in the event select field of the ESCR to select the event.

ESCR Event Mask

Gives the value to be placed in the Event Mask field of the ESCR to select sub-events to be counted. The parameter value column defines the documented bits with relative bit position offset starting from 0 (where the absolute bit position of relative offset 0 is bit 9 of the ESCR. All undocumented bits are reserved and should be set to 0.

CCCR Select Gives the value to be placed in the ESCR select field of the CCCR associated with the counter to select the ESCR to be used to define the event. (Note that this value is not the address of the ESCR; instead, it is the number of the ESCR from the Number column in Table 15-4.)

Event Specific Notes
Gives additional information about the event, such as the name of the same or a similar event defined for the P6 family processors.

Can Support PEBS
Indicates if PEBS is supported for the event. (This information is only supplied for at-retirement events listed in Table A-2.)

Requires Additional MSR for Tagging
Indicates which if any additional MSRs must be programmed to count the events. (This information is only supplied for the at-retirement events listed in Table A-2.)

NOTE

The performance-monitoring events listed in Appendix A, *Performance-Monitoring Events* are intended to be used as guides for performance tuning. The counter values reported are not guaranteed to be absolutely accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

The following procedure shows how to set up a performance counter for basic counting; that is, the counter is set up to count a specified event indefinitely, wrapping around whenever it reaches its maximum count. This procedure is continued through the following four sections.

Using the information given in Table A-1, an event to be counted can be selected as follows:

1. Select the event to be counted.
2. Select the ESCR to be used to select events to be counted from the ESCRs field.
3. Select the number of the counter to be used to count the event from the Counter Numbers Per ESCR field.
4. Determine the name of the counter and the CCCR associated with the counter, and determine the MSR addresses of the counter, CCCR, and ESCR from Table 15-4.
5. Use the WRMSR instruction to write the ESCR Event Select and ESCR Event Mask values from Table A-1 into the appropriate fields in the ESCR. At the same time set or clear the USR and OS flags in the ESCR as desired.
6. Use the WRMSR instruction to write the CCCR Select value from Table A-1 into the appropriate field in the CCCR.

NOTE

Typically all the fields and flags of the CCCR will be written with one WRMSR instruction; however, in this procedure, several WRMSR writes are used to more clearly demonstrate the uses of the various CCCR fields and flags.

This setup procedure is continued in the next section, Section 15.9.6.2., “Filtering Events”.

15.9.6.2. FILTERING EVENTS

Each counter receives up to 4 input lines from the processor hardware from which it is counting events. The counter treats these inputs as binary inputs (input 0 has a value of 1, input 1 has a value of 2, input 3 has a value of 4, and input 3 has a value of 8). When a counter is enabled, it adds this binary input value to the counter value on each clock cycle. For each clock cycle, the value added to the counter can then range from 0 (no event) to 15.

For many events, only the 0 input line is active, so the counter is merely counting the clock cycles during which the 0 input is asserted. However, for some events two or more input lines are used. Here, the counters threshold setting can be used to filter events. The compare, complement, threshold, and edge fields control the filtering of counter increments by input value.

If the compare flag is set, then a “greater than” or a “less than or equal to” comparison of the input value vs. a threshold value can be made. The complement flag selects “less than or equal to” (flag set) or “greater than” (flag clear). The threshold field selects a threshold value of from 0 to 15. For example, if the complement flag is cleared and the threshold field is set to 6, than any input value of 7 or greater on the 4 inputs to the counter will cause the counter to be incremented by 1, and any value less than 7 will cause an increment of 0 (or no increment) of the counter. Conversely, if the complement flag is set, any value from 0 to 6 will increment the counter and any value from 7 to 15 will not increment the counter. Note that when a threshold condition has been satisfied, the input to the counter is always 1, not the input value that is presented to the threshold filter.

The edge flag provides further filtering of the counter inputs when a threshold comparison is being made. The edge flag is only active when the compare flag is set. When the edge flag is set, the resulting output from the threshold filter (a value of 0 or 1) is used as an input to the edge filter. Each clock cycle, the edge filter examines the last and current input values and sends a count to the counter only when it detects a “rising edge” event; that is, a false-to-true transition. Figure 15-14 illustrates rising edge filtering.

The following procedure shows how to configure a CCCR to filter events using the threshold filter and the edge filter. This procedure is a continuation of the setup procedure introduced in Section 15.9.6.1., “Selecting Events to Count”.

7. (Optional) To set up the counter for threshold filtering, use the WRMSR instruction to write values in the CCCR compare and complement flags and the threshold field:
   — Set the compare flag.
   — Set or clear the complement flag for less than or equal to or greater than comparisons, respectively.
   — Enter a value from 0 to 15 in the threshold field.
8. (Optional) Select rising edge filtering by setting the CCCR edge flag.

This setup procedure is continued in the next section, Section 15.9.6.3., "Starting Event Counting".

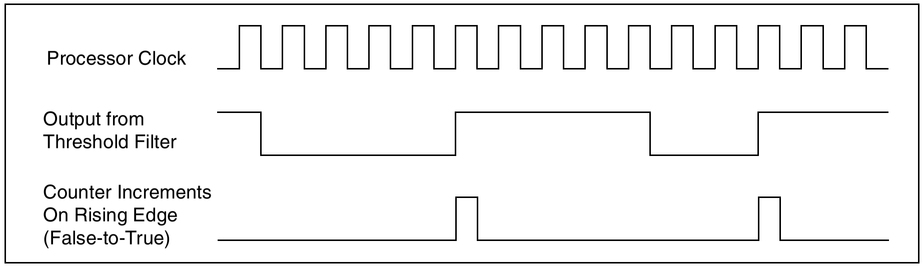

Figure 15-14. Effects of Edge Filtering

15.9.6.3. STARTING EVENT COUNTING

Event counting by a performance counter can be initiated in either of two ways. The typical way is to set the enable flag in the counter's CCCR. Following the instruction to set the enable flag, event counting begins and continues until it is stopped (see Section 15.9.6.5., "Halting Event Counting").

The following procedural step shows how to start event counting. This step is a continuation of the setup procedure introduced in Section 15.9.6.2., "Filtering Events".

9. To start event counting, use the WRMSR instruction to set the CCCR enable flag for the performance counter.

This setup procedure is continued in the next section, Section 15.9.6.4., "Reading a Performance Counter's Count".

The second way that a counter can be started by using the cascade feature. Here, the overflow of one counter automatically starts its alternate counter (see Section 15.9.6.6., "Cascading Counters").

15.9.6.4. READING A PERFORMANCE COUNTER'S COUNT

The Pentium 4 and Intel Xeon processors' performance counters can be read using either the RDPMC or RDMSR instructions. The enhanced functions of the RDPMC instruction (including fast read) are described in Section 15.9.2., "Performance Counters". These instructions can be used to read a performance counter while it is counting or when it is stopped.

The following procedural step shows how to read the event counter. This step is a continuation of the setup procedure introduced in Section 15.9.6.3., "Starting Event Counting".

10. To read a performance counters current event count, execute the RDPMC instruction with the counter number obtained from Table 15-4 used as an operand.

This setup procedure is continued in the next section, Section 15.9.6.5., "Halting Event Counting".

15.9.6.5. HALTING EVENT COUNTING

After a performance counter has been started (enabled), it continues counting indefinitely. If the counter overflows (goes one count past its maximum count), it wraps around and continues counting. When the counter wraps around, it sets its OVF flag to indicate that the counter has overflowed. The OVF flag is a sticky flag that indicates that the counter has overflowed at least once since the OVF bit was last cleared.

To halt counting, the CCCR enable flag for the counter must be cleared.

The following procedural step shows how to stop event counting. This step is a continuation of the setup procedure introduced in Section 15.9.6.4., "Reading a Performance Counter's Count".

11. To stop event counting, execute a WRMSR instruction to clear the CCCR enable flag for the performance counter.

To halt a cascaded counter (a counter that was started when its alternate counter overflowed), either clear the Cascade flag in the cascaded counter's CCCR MSR or clear the OVF flag in the alternate counter's CCCR MSR.

15.9.6.6. CASCADING COUNTERS

As described in Section 15.9.2., "Performance Counters", eighteen performance counters are implemented in pairs. Nine pairs of counters and associated CCCRs are further organized as four blocks: BPU, MS, FLAME, and IQ (see Table 15-4). The first three blocks contain two pairs each. The IQ block contains three pairs of counters (12 through 17) with associated CCCRs (MSR_IQ_CCCR0 through MSR_IQ_CCCR5).

The first 8 counter pairs (0 through 15) can be programmed using ESCRs to detect performance monitoring events. Pairs of ESCRs in each of the four blocks allow many different types of events to be counted. The cascade flag in the CCCR MSR allows nested monitoring of events to be performed by cascading one counter to a second counter located in another pair in the same block (see Figure 15-9 for the location of the flag).

Counters 0 and 1 form the first pair in the BPU block. Either counter 0 or 1 can be programmed to detect an event via MSR_MO B_ESCR0. Counters 0 and 2 can be cascaded in any order, as can counters 1 and 3. It's possible to set up 4 counters in the same block to cascade on two pairs of independent events. The pairing described also applies to subsequent blocks. Since the IQ PUB has two extra counters, cascading operates somewhat differently if 16 and 17 are involved. In the IQ block, counter 16 can only be cascaded from counter 14 (not from 12); counter 14 cannot be cascaded from counter 16 using the CCCR cascade bit mechanism. Similar restrictions apply to counter 17.

Example Scenario

Assume a scenario where counter X is set up to count 200 occurrences of event A; then counter Y is set up to count 400 occurrences of event B. Each counter is set up to count a specific event and overflow to the next counter. In the above example, counter X is preset for a count of -200 and counter Y for a count of -400; this setup causes the counters to overflow on the 200th and 400th counts respectively.

Continuing this scenario, counter X is set up to count indefinitely and wraparound on overflow (as described in the basic performance counter setup procedure that begins in Section 15.9.6.1., "Selecting Events to Count"). Counter Y is set up with the cascade flag in its associated CCCR MSR set to 1 and its enable flag set to 0.

To begin the nested counting, the enable bit for the counter X is set. Once enabled, counter X counts until it overflows. At this point, counter Y is automatically enabled and begins counting. Thus counter X overflows after 200 occurrences of event A. Counter Y then starts, counting 400 occurrences of event B before overflowing. When performance counters are cascaded, the counter Y would typically be set up to generate an interrupt on overflow, as described in Section 15.9.6.9., "Generating an Interrupt on Overflow".

The cascading counters mechanism can be used to count a single event. The counting begins on one counter then continues on the second counter after the first counter overflows. This technique doubles the number of event counts that can be recorded, since the contents of the two counters can be added together.

15.9.6.7. EXTENDED CASCADING

Extended cascading is a model-specific feature in the Intel NetBurst microarchitecture. The feature is available to Pentium 4 and Xeon processor family with family encoding of 15 and model encoding greater than or equal to 2. This feature uses bit 11 in CCCRs associated with the IQ block. See the table below.

Table 15-5. CCR Names and Bit Positions

CCCR Name:Bit Position	Bit Name	Description
MSR_IQ_CCCR1|2:11	Reserved	
MSR_IQ_CCCR0:11	CASCNT4INTO0	Allow counter 4 to cascade into counter 0
MSR_IQ_CCCR3:11	CASCNT5INTO3	Allow counter 5 to cascade into counter 3
MSR_IQ_CCCR4:11	CASCNT5INTO4	Allow counter 5 to cascade into counter 4
MSR_IQ_CCCR5:11	CASCNT4INTO5	Allow counter 4 to cascade into counter 5

The extended cascading feature can be adapted to the sampling usage model for performance monitoring. However, it is known that performance counters do not generate PMI in cascade mode or extended cascade mode due to an erratum. This erratum applies to Pentium 4 and Intel Xeon processors with model encoding of 2. For Pentium 4 and Intel Xeon processors with model encoding of 0 and 1, the erratum applies to processors with stepping encoding greater than 09H.

15.9.6.8. EXTENDED CASCADING

Counters 16 and 17 in the IQ block are frequently used in precise event-based sampling or at-retirement counting of events indicating a stalled condition in the pipeline. Neither counter 16 or 17 can initiate the cascading of counter pairs using the cascade bit in a CCCR.

Extended cascading permits performance monitoring tools to use counters 16 and 17 to initiate cascading of two counters in the IQ block. Extended cascading from counter 16 and 17 is conceptually similar to cascading other counters, but instead of using CASCADE bit of a CCCR, one of the four CASCNTxINTOy bits is used.

Example Scenario

A usage scenario for extended cascading is to sample instructions retired on logical processor 1 after the first 4096 instructions retired on logical processor 0. A procedure to program extended cascading in this scenario is outlined below:

1. Write the value 0 to counter 12.
2. Write the value 04000603H to MSR_CRU_ESCR0 (corresponding to selecting the NBOGNTAG and NBOGTAG event masks with qualification restricted to logical processor 1.).
1. Write the value 04038800H to MSR_IQ_CCCR0 to enable CASCNT4INTO0, and OVF_PMI. An ISR can sample on instruction addresses in this case. (Do not set ENABLE, nor CASCADE).
2. Write the value FFFFF000H into counter 16.

3. Write the value 0400060CH to MSR_CRU_ESCR2 (corresponding to selecting the NBOGNTAG and NBOGTAG event masks with qualification restricted to logical processor 0).
4. Write the value 00039000H to MSR_IQ_CCCR4 (set ENABLE bit, but not OVF_PMI).

Another use for cascading is to locate stalled execution in a multithreaded application. Assume MOB replays in thread B cause thread A to stall. Getting a sample of the stalled execution in this scenario could be accomplished by:

1. Set up counter B to count MOB replays on thread B.
2. Set up counter A to count resource stalls on thread A; set its force overflow bit and the appropriate CASCNTxINTOy bit.
3. Use the performance monitoring interrupt to capture the program execution data of the stalled thread.

15.9.6.9. GENERATING AN INTERRUPT ON OVERFLOW

Any of the performance counters can be configured to generate a performance monitor interrupt (PMI) if the counter overflows. The PMI interrupt service routine can then collect information about the state of the processor or program when overflow occurred. This information can then be used with a tool like the VTune™ Performance Analyzer to analyze and tune program performance.

To enable an interrupt on counter overflow, the OVR_PMI flag in the counter's associated CCCR MSR must be set. When overflow occurs, the PMI is generated through the local APIC. (Here, the performance counter entry in the local vector table [LVT] is set up to deliver the interrupt generated by the PMI to the processor.)

The PMI service routine can use the OVF flag to determine which counter overflowed when multiple counters have been configured generate PMIs.

When generating interrupts on overflow, the performance counter being used should be preset to value that will cause an overflow after a specified number of events are counted plus 1. The simplest way to select the preset value is to write a negative number into the counter, as described in Section 15.9.6.6., "Cascading Counters". Here, however, if an interrupt is to be generated after 100 event counts, the counter should be preset to minus 100 plus 1 (-100 + 1), or -99. The counter will then overflow after it counts 99 events and generate an interrupt on the next (100th) event counted. The difference of 1 for this count enables the interrupt to be generated immediately after the selected event count has been reached, instead of waiting for the overflow to be propagation through the counter.

Because of latency in the micro-architecture between the generation of events and the generation of interrupts on overflow, it is sometimes difficult to generate an interrupt close to an event that caused it. In these situations, the FORCE_OVF flag in the CCCR can be used to improve reporting. Setting this flag causes the counter to overflow on every counter increment, which in turn triggers an interrupt after every counter increment.

15.9.6.10. COUNTER USAGE GUIDELINE

There are some instances where the user must take care to configure counting logic properly, so that it is not powered down. To use any ESCR, even when it is being used just for tagging, (any) one of the counters that the particular ESCR (or its paired ESCR) can be connected to should be enabled. If this is not done, 0 counts may result. Likewise, to use any counter, there must be some event selected in a corresponding ESCR (other than no_event, which generally has a select value of 0).

15.9.7. At-Retirement Counting

At-retirement counting provides a means counting only events that represent work committed to architectural state and ignoring work that was performed speculatively and later discarded.

The Intel NetBurst micro-architecture used in the Pentium 4 and Intel Xeon processors performs many speculative activities in an attempt to increase effective processing speeds. One example of this speculative activity is branch prediction. The Pentium 4 and Intel Xeon processors typically predict the direction of branches and then decode and execute instructions down the predicted path in anticipation of the actual branch decision. When a branch misprediction occurs, the results of instructions that were decoded and executed down the mispredicted path are canceled. If a performance counter was set up to count all executed instructions, the count would include instructions whose results were canceled as well as those whose results committed to architectural state.

To provide finer granularity in event counting in these situations, the performance monitoring facilities provided in the Pentium 4 and Intel Xeon processors provide a mechanism for tagging events and then counting only those tagged events that represent committed results. This mechanism is called "at-retirement counting."

Tables A-2 through A-5 list predefined at-retirement events and event metrics that can be used to for tagging events when using at retirement counting. The following terminology is used in describing at-retirement counting.

Bogus, Non-Bogus, Retire

In at-retirement event descriptions, the term "bogus" refers to instructions or µops that must be canceled because they are on a path taken from a mispredicted branch. The terms "retired" and "non-bogus" refer to instructions or µops along the path that results in committed architectural state changes as required by the program being executed. Thus instructions and µops are either bogus or non-bogus, but not both. Several of the Pentium 4 and Intel Xeon processors' performance monitoring events (such as, Instruction_Retired and Uops_Retired in Table A-2) can count instructions or µops that are retired based on the characterization of bogus" versus non-bogus.

Tagging Tagging is a means of marking µops that have encountered a particular performance event so they can be counted at retirement. During the course of execution, the same event can happen more than once per µop and a direct count of the event would not provide an indication of how many µops encountered that event.

The tagging mechanisms allow a µop to be tagged once during its lifetime and thus counted once at retirement. The retired suffix is used for performance metrics that increment a count once per µop, rather than once per event. For example, a µop may encounter a cache miss more than once during its life time, but a "Miss Retired" metric (that counts the number of retired µops that encountered a cache miss) will increment only once for that µop. A "Miss Retired" metric would be useful for characterizing the performance of the cache hierarchy for a particular instruction sequence. Details of various performance metrics and how these can be constructed using the Pentium 4 and Intel Xeon processors performance events are provided in the *Intel Pentium 4 Processor Optimization Reference Manual* (see Section 1.4., "Related Literature").

Replay To maximize performance for the common case, the Intel NetBurst microarchitecture aggressively schedules µops for execution before all the conditions for correct execution are guaranteed to be satisfied. In the event that all of these conditions are not satisfied, µops must be reissued. The mechanism that the Pentium 4 and Intel Xeon processors use for this reissuing of µops is called replay. Some examples of replay causes are cache misses, dependence violations, and unforeseen resource constraints. In normal operation, some number of replays is common and unavoidable. An excessive number of replays is an indication of a performance problem.

Assist When the hardware needs the assistance of microcode to deal with some event, the machine takes an assist. One example of this is an underflow condition in the input operands of a floating-point operation. The hardware must internally modify the format of the operands in order to perform the computation. Assists clear the entire machine of µops before they begin and are costly.

15.9.7.1. USING AT-RETIREMENT COUNTING

The Pentium 4 and Intel Xeon processors allow counting both events and µops that encountered a specified event. For a subset of the at-retirement events listed in Table A-2, a µop may be tagged when it encounters that event. The tagging mechanisms can be used in non-precise event-based sampling, and a subset of these mechanisms can be used in PEBS. There are four independent tagging mechanisms, and each mechanism uses a different event to count µops tagged with that mechanism:

- **Front-end tagging.** This mechanism pertains to the tagging of µops that encountered front-end events (for example, trace cache and instruction counts) and are counted with the Front_end_event event
- **Execution tagging.** This mechanism pertains to the tagging of µops that encountered execution events (for example, instruction types) and are counted with the Execution_Event event.
- **Replay tagging.** This mechanism pertains to tagging of µops whose retirement is replayed (for example, a cache miss) and are counted with the Replay_event event. Branch mispredictions are also tagged with this mechanism.

- **No tags.** This mechanism does not use tags. It uses the Instr_retired and the Uops_ retired events.

Each tagging mechanism is independent from all others; that is, a µop that has been tagged using one mechanism will not be detected with another mechanism's tagged-µop detector. For example, if µops are tagged using the front-end tagging mechanisms, the Replay_event will not count those as tagged µops unless they are also tagged using the replay tagging mechanism. However, execution tags allow up to four different types of µops to be counted at retirement through execution tagging.

The independence of tagging mechanisms does not hold when using PEBS. When using PEBS, only one tagging mechanism should be used at a time.

Certain kinds of µops that cannot be tagged, including I/O, uncacheable and locked accesses, returns, and far transfers.

Table A-2 lists the performance monitoring events that support at-retirement counting: specifically the Front_end_event, Execution_event, Replay_event, Inst_retired and Uops_retired events. The following sections describe the tagging mechanisms for using these events to tag µop and count tagged µops.

15.9.7.2. TAGGING MECHANISM FOR FRONT_END_EVENT

The Front_end_event counts µops that have been tagged as encountering any of the following events:

- **µop decode events.** Tagging µops for µop decode events requires specifying bits in the ESCR associated with the performance-monitoring event, Uop_type.
- **Trace cache events.** Tagging µops for trace cache events may require specifying certain bits in the MSR_TC_PRECISE_EVENT MSR (see Table A-3).

Table A-2 describes the Front_end_event and Table A-3 describes metrics that are used to set up a Front_end_event count.

The MSRs specified in the Table A-2 that are supported by the front-end tagging mechanism must be set and one or both of the NBOGUS and BOGUS bits in the Front_end_event event mask must be set to count events. None of the events currently supported requires the use of the MSR_TC_PRECISE_EVENT MSR.

15.9.7.3. TAGGING MECHANISM FOR EXECUTION_EVENT

Table A-2 describes the Execution_event and Table A-4 describes metrics that are used to set up an Execution_event count.

The execution tagging mechanism differs from other tagging mechanisms in how it causes tagging. One *upstream* ESCR is used to specify an event to detect and to specify a tag value (bits 5 through 8) to identify that event. A second *downstream* ESCR is used to detect µops that have been tagged with that tag value identifier using Execution_event for the event selection.

The upstream ESCR that counts the event must have its tag enable flag (bit 4) set and must have an appropriate tag value mask entered in its tag value field. The 4-bit tag value mask specifies which of tag bits should be set for a particular µop. The value selected for the tag value should coincide with the event mask selected in the downstream ESCR. For example, if a tag value of 1 is set, then the event mask of NBOGUS0 should be enabled, correspondingly in the downstream ESCR. The downstream ESCR detects and counts tagged µops. The normal (not tag value) mask bits in the downstream ESCR specify which tag bits to count. If any one of the tag bits selected by the mask is set, the related counter is incremented by one. This mechanism is summarized in the Table A-4 metrics that are supported by the execution tagging mechanism. The tag enable and tag value bits are irrelevant for the downstream ESCR used to select the Execution_event.

The four separate tag bits allow the user to simultaneously but distinctly count up to four execution events at retirement (This applies for non-precise event-based sampling. There are additional restrictions for PEBS as noted in Section 15.9.8.3., "Setting Up the PEBS Buffer"). It is also possible to detect or count combinations of events by setting multiple tag value bits in the upstream ESCR or multiple mask bits in the downstream ESCR. For example, use a tag value of 3H in the upstream ESCR and use NBOGUS0/NBOGUS1 in the downstream ESCR event mask.

15.9.7.4. TAGGING MECHANISM FOR REPLAY_EVENT

Table A-2 describes the Replay_event and Table A-5 describes metrics that are used to set up an Replay_event count.

The replay mechanism enables tagging of µops for a subset of all replays before retirement. Use of the replay mechanism requires selecting the type of µop that may experience the replay in the MSR_PEBS_MATRIX_VERT MSR and selecting the type of event in the IA32_PEBS_ENABLE MSR. Replay tagging must also be enabled with the UOP_Tag flag (bit 24) in the IA32_PEBS_ENABLE MSR.

The Table A-5 lists the metrics that are support the replay tagging mechanism and the at-retirement events that use the replay tagging mechanism, and specifies how the appropriate MSRs need to be configured. The replay tags defined in Table A-5 also enable Precise Event-Based Sampling (PEBS, see Section 15.9.8). Each of these replay tags can also be used in normal sampling by not setting Bit 24 nor Bit 25 in IA_32_PEBS_ENABLE_MSR. Each of these metrics requires that the Replay_Event (see Table A-2) be used to count the tagged µops.

15.9.8. Precise Event-Based Sampling (PEBS)

The debug store (DS) mechanism in the Pentium 4 and Intel Xeon processors allow two types of information to be collected for use in debugging and tuning programs: PEBS records and BTS records. (See Section 15.5.7., "Branch Trace Store (BTS)" for a description of the BTS mechanism.)

PEBS permits the saving of precise architectural information associated with one or more performance events in the precise event records buffer, which is part of the DS save area (see Section 15.9.5., "DS Save Area"). To use this mechanism, a counter is configured to overflow after it has counted a preset number of events. When the counter overflows, the processor copies the current state of the general-purpose and EFLAGS registers and instruction pointer into a record in the precise event records buffer. The processor then resets the count in the performance counter and restarts the counter. When the precise event records buffer is nearly full, an interrupt is generated, allowing the precise event records to be saved. A circular buffer is not supported for precise event records.

PEBS is supported only for a subset of the at-retirement events: Execution_event, Front_end_event, and Replay_event. Also, PEBS can only carried out using the one performance counter, the MSR_IQ_COUNTER4 MSR.

15.9.8.1. DETECTION OF THE AVAILABILITY OF THE PEBS FACILITIES

The DS feature flag (bit 21) returned by the CPUID instruction indicates (when set) the availability of the DS mechanism in the processor, which supports the PEBS (and BTS) facilities. When this bit is set, the following PEBS facilities are available:

- The PEBS_UNAVAILABLE flag in the IA32_MISC_ENABLE MSR indicates (when clear) the availability of the PEBS facilities, including the IA32_PEBS_ENABLE MSR.
- The enable PEBS flag (bit 24) in the IA32_PEBS_ENABLE MSR allows PEBS to be enabled (set) or disabled (clear).
- The IA32_DS_AREA MSR can be programmed to point to the DS save area.

15.9.8.2. SETTING UP THE DS SAVE AREA

Section 15.5.7.2., "Setting Up the DS Save Area" describes how to set up and enable the DS save area. This procedure is common for PEBS and BTS:

15.9.8.3. SETTING UP THE PEBS BUFFER

Only the MSR_IQ_COUNTER4 performance counter can be used for PEBS. Use the following procedure to set up the processor and this counter for PEBS:

1. Set up the precise event buffering facilities. Place values in the precise event buffer base, precise event index, precise event absolute maximum, and precise event interrupt threshold, and precise event counter reset fields of the DS buffer management area (see Figure 15-10) to set up the precise event records buffer in memory.
2. Enable PEBS. Set the Enable PEBS flag (bit 24) in IA32_PEBS_ENABLE MSR.
3. Set up the MSR_IQ_COUNTER4 performance counter and its associated CCCR and one or more ESCRs for PEBS as described in Tables A-2 through A-5.

15.9.8.4. WRITING A PEBS INTERRUPT SERVICE ROUTINE

The PEBS facilities share the same interrupt vector and interrupt service routine (called the DS ISR) with the non-precise event-based sampling and BTS facilities. To handle PEBS interrupts, PEBS handler code must be included in the DS ISR. See Section 15.5.7.4., "Writing the DS Interrupt Service Routine" for guidelines for writing the DS ISR.

15.9.8.5. OTHER DS MECHANISM IMPLICATIONS

The DS mechanism is not available in the SMM. It is disabled on transition to the SMM mode. Similarly the DS mechanism is disabled on the generation of a machine check exception and is cleared on processor RESET and INIT. The DS mechanism is available in real address mode.

15.9.9. Counting Clocks

The count of cycles, also known as clockticks, forms a the basis for measuring how long a program takes to execute. Clockticks are also used as part of efficiency ratios like cycles per instruction (CPI). Processor clocks may stop ticking under circumstances like the following:

- The processor is halted when there is nothing for the CPU to do. For example, the processor may halt to save power while the computer is servicing an I/O request. When Hyper-Threading Technology is enabled, both logical processors must be halted for performance-monitoring counters to be powered down.
- The processor is asleep as a result of being halted or because of a power-management scheme. There are different levels of sleep. In the some deep sleep levels, the Time Stamp Counter stops counting.

There are three ways to count processor clock cycles to monitor performance. These are:

- Non-Halted Clockticks – Measures clock cycles in which the specified logical processor is not halted and is not in any power-saving state. When Hyper-Threading Technology is enabled, this these ticks can be measured on a per-logical-processor basis.
- Non-Sleep Clockticks – Measures clock cycles in which the specified physical processor is not in a sleep mode or in a power-saving state. These ticks can**not** be measured on a logical-processor basis.
- Time Stamp Counter – Measures clock cycles in which the physical processor is not in deep sleep. These ticks can**not** be measured on a logical-processor basis.

The first two methods use performance counters and can be set up to cause an interrupt upon overflow (for sampling). They may also be useful where it is easier for a tool to read a performance counter than to use a time stamp counter (the timestamp counter is accessed via an instruction, RDTSC).

For applications with a significant amount of I/O, there are two ratios of interest:

- Non-Halted CPI: Non-halted clockticks/instructions retired measures the CPI for phases where the CPU was being used. This ratio can be measured on a logical-processor basis when Hyper-Threading Technology is enabled.
- Nominal CPI: Time stamp counter ticks/instructions retired measures the CPI over the duration of a program, including those periods when the machine halts while waiting for I/O.

Non-Halted Clockticks

Use the following procedure to program ESCRs and CCCRs to obtain non-halted clock ticks:

1. Select an ESCR for the global_power_events and specify the RUNNING sub-event mask and the desired T0_OS/T0_USR/T1_OS/T1_USR bits for the targeted processor.
2. Select an appropriate counter.
3. Enable counting in the CCCR for that counter by setting the enable bit.

Non-Sleep Clockticks

Performance monitoring counters can be configured to count clockticks whenever the performance monitoring hardware is not powered-down. To count Non-sleep Clockticks with a performance-monitoring counter, do the following:

1. Select one of the 18 counters.
2. Select any of the ESCRs whose events the selected counter can count. Set its event select to anything other than no_event. This may not seem necessary, but the counter may be disabled if this is not done.
3. Turn threshold comparison on in the CCCR by setting the compare bit to 1.
4. Set the threshold to 15 and the complement to 1 in the CCCR. Since no event can exceed this threshold, the threshold condition is met every cycle and the counter counts every cycle. Note that this overrides any qualification (e.g. by CPL) specified in the ESCR.
5. Enable counting in the CCCR for the counter by setting the enable bit.

In most cases, the counts produced by the non-halted and non-sleep metrics are equivalent if the physical package supports one logical processor and is not placed in a power-saving state. Operating systems may execute an HLT instruction and place a physical processor in a power-saving state.

On processors that support Hyper-Threading Technology (HT), each physical package can support two or more logical processors. Current implementation of HT provides two logical processors for each physical processor. While both logical processors can execute two threads simultaneously, one logical processor may halt to allow the other logical processor to execute without sharing execution resources between two logical processors.

Non-halted Clockticks can be set up to count the number of processor clock cycles for each logical processor whenever the logical processor is not halted (the count may include some portion of the clock cycles for that logical processor to complete a transition to a halted state). Physical processors that support HT enter into a power-saving state if all logical processors halt.

The Non-sleep Clockticks mechanism uses a filtering mechanism in CCCRs. The mechanism will continue to increment as long as one logical processor is not halted or in a power-saving state. Applications may cause a processor to enter into a power-saving state by using an OS service that transfers control to an OS's idle loop. The idle loop then may place the processor into a power-saving state after an implementation-dependent period if there is no work for the processor.

Time Stamp Counter

The Time Stamp Counter increments whenever the sleep pin is not asserted or when the clock signal on the system bus is active. The counter can be read with the RDTSC instruction. Computing the difference in values between two reads (modulo 2^{64}) gives the number of processor clocks between reads.

The Time Stamp Counter and the Non-sleep Clockticks count should agree in practically all cases. However, it is possible to have both logical processors in a physical package halted, resulting in most of the chip (including the performance monitoring hardware) being powered down. When this happens, it is possible for the time stamp counter to continue incrementing because the clock signal on the system bus is still active. Non-sleep Clockticks will no longer increment because the performance monitoring hardware is powered down in a power-saving state.

15.9.10. Operating System Implications

The DS mechanism can be used by the operating system as a debugging extension to facilitate failure analysis. When using this facility, a 25 to 30 times slowdown can be expected due to the effects of the trace store occurring on every taken branch.

Depending upon intended usage, the instruction pointers that are part of the branch records or the PEBS records need to have an association with the corresponding process. One solution requires the ability for the DS specific operating system module to be chained to the context switch. A separate buffer can then be maintained for each process of interest and the MSR pointing to the configuration area saved and setup appropriately on each context switch.

If the BTS facility has been enabled, then it must be disabled and state stored on transition of the system to a sleep state in which processor context is lost. The state must be restored on return from the sleep state.

It is required that an interrupt gate be used for the DS interrupt as opposed to a trap gate to prevent the generation of an endless interrupt loop.

Pages that contain buffers must have mappings to the same physical address for all processes/logical processors, such that any change to CR3 will not change DS addresses. If this requirement cannot be satisfied (that is, the feature is enabled on a per thread/process basis), then the operating system must ensure that the feature is enabled/disabled appropriately in the context switch code.

15.10. PERFORMANCE MONITORING AND HYPER-THREADING TECHNOLOGY

The performance monitoring capability of IA-32 processors with Hyper-Threading Technology is similar to that on the Pentium 4 and Intel Xeon processors. However, the performance monitoring capability is extended so that:

- The performance counters can be programmed to select events that are qualified by logical processor IDs.
- Performance monitoring interrupts can be directed to a specific logical processor within the physical processor.

This section describes the programming interfaces with respect to using performance counters, qualifying events by logical processor IDs, additional programmable bits in ESCRs, and CCCRs, as well as the special purpose IA32_PEBS_ENABLE, MSR_PEBS_MATRIX_VERT, and MSR_TC_PRECISE_EVENT MSRs.

In the Intel IA-32 processors with Hyper-Threading Technology, these registers are shared between the two logical processors in the physical processor. To allow these shared registers to be used to monitor performance events on either logical processor or both, additional flags have been added to the ESCR and CCCR MSRs and to the IA32_PEBS_ENABLE MSR. These additional flags and the effect of these flags on event monitoring in while Hyper-Threading Technology is active are described in the following sections.

15.10.1. ESCR MSRs

Figure 15-15 shows the layout of an ESCR MSR in the Intel IA-32 processors with Hyper Threading Technology.

The functions of the flags and fields are as follows:

T1_USR flag, bit 0

When set, events are counted when thread 1 (logical processor 1) is executing at a current privilege level (CPL) of 1, 2, or 3. These privilege levels are generally used by application code and unprotected operating system code.

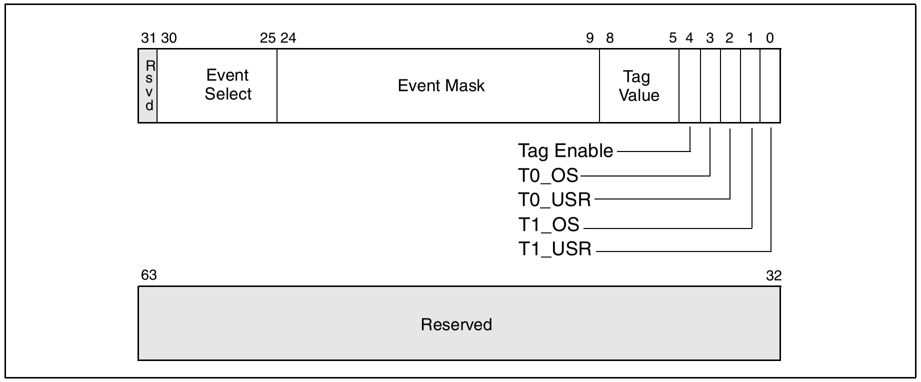

Figure 15-15. Event Selection Control Register (ESCR) for Intel Xeon processor MP

T1_OS flag, bit 1
When set, events are counted when thread 1 (logical processor 1) is executing at CPL of 0. This privilege level is generally reserved for protected operating system code. (When both the T1_OS and T1_USR flags are set, thread 1 events are counted at all privilege levels.)

T0_USR flag, bit 2
When set, events are counted when thread 0 (logical processor 0) is executing at a CPL of 1, 2, or 3.

T0_OS flag, bit 3
When set, events are counted when thread 0 (logical processor 0) is executing at CPL of 0. (When both the T0_OS and T0_USR flags are set, thread 0 events are counted at all privilege levels.)

Tag Enable, bit 4
When set, enables tagging of µops to assist in at-retirement event counting; when clear, disables tagging. See Section 15.9.7., "At-Retirement Counting".

Tag Value field, bits 5 through 8
Selects a tag value to associate with a µop to assist in at-retirement event counting.

Event Mask field, bits 9 through 24
Selects events to be counted from the event class selected with the event select field.

Event Select field, bits 25 through 30)
Selects a class of events to be counted. The events within this class that are counted are selected with the event mask field.

The T0_OS and T0_USR flags and the T1_OS and T1_USR flags allow event counting and sampling to be specified for a specific logical processor (0 or 1) within an Intel Xeon processor

MP. (See Section 7.6.8., "Identifying Logical Processors in an MP System" for information on identifying logical processor in an Xeon processor MP.)

Not all performance monitoring events can be detected within an Intel Xeon processor MP on a per logical processor basis (see Section 15.10.4., "Performance Monitoring Events"). Some sub-events (specified by an event mask bits) are counted or sampled without regard to which logical processor is associated with the detected event.

15.10.2. CCCR MSRs

Figure 15-16 shows the layout of a CCCR MSR in the Intel IA-32 processors with Hyper-Threading Technology.

The functions of the flags and fields are as follows:

Enable flag, bit 12
When set, enables counting; when clear, the counter is disabled. This flag is cleared on reset

ESCR Select field, bits 13 through 15
Identifies the ESCR to be used to select events to be counted with the counter associated with the CCCR.

Active Thread field, bits 16 and 17
Enables counting depending on which logical processors are active (executing a thread). This field enables filtering of events based on the state (active or inactive) of the logical processors. The encodings of this field are as follows:

00—None. Count only when neither logical processor is active.

01—Single. Count only when one logical processor is active (either 0 or 1).

10—Both. Count only when both logical processors are active.

11—Any. Count when either logical processor is active.

Note that a halted logical processor or a logical processor in the "wait for SIPI" state is considered inactive.

Compare flag, bit 18
When set, enables filtering of the event count; when clear, disables filtering. The filtering method is selected with the threshold, complement, and edge flags.

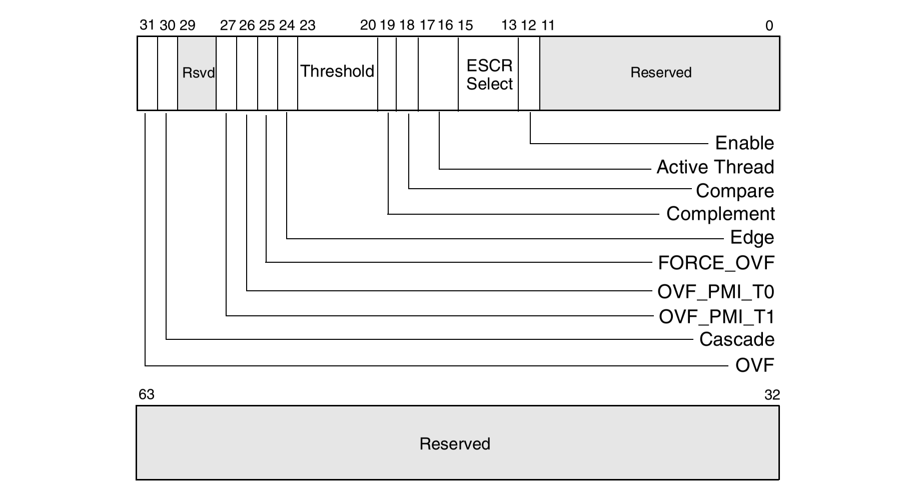

Figure 15-16. Counter Configuration Control Register (CCCR)

Complement flag, bit 19

Selects how the incoming event count is compared with the threshold value. When set, event counts that are less than or equal to the threshold value result in a single count being delivered to the performance counter; when clear, counts greater than the threshold value result in a count being delivered to the performance counter (see Section 15.9.6.2., "Filtering Events"). The compare flag is not active unless the compare flag is set.

Threshold field, bits 20 through 23

Selects the threshold value to be used for comparisons. The processor examines this field only when the compare flag is set, and uses the complement flag setting to determine the type of threshold comparison to be made. The useful range of values that can be entered in this field depend on the type of event being counted (see Section 15.9.6.2., "Filtering Events").

Edge flag, bit 24

When set, enables rising edge (false-to-true) edge detection of the threshold comparison output for filtering event counts; when clear, rising edge detection is disabled. This flag is active only when the compare flag is set.

FORCE_OVF flag, bit 25

When set, forces a counter overflow on every counter increment; when clear, overflow only occurs when the counter actually overflows.

OVF_PMI_T0 flag, bit 26

When set, causes a performance monitor interrupt (PMI) to be sent to logical processor 0 when the counter overflows occurs; when clear, disables PMI

generation for logical processor 0. Note that the PMI is generate on the next event count after the counter has overflowed.

OVF_PMI_T1 flag, bit 27

When set, causes a performance monitor interrupt (PMI) to be sent to logical processor 1 when the counter overflows occurs; when clear, disables PMI generation for logical processor 1. Note that the PMI is generate on the next event count after the counter has overflowed.

Cascade flag, bit 30

When set, enables counting on one counter of a counter pair when its alternate counter in the other the counter pair in the same counter group overflows (see Section 15.9.2., "Performance Counters" for further details); when clear, disables cascading of counters.

OVF flag, bit 31

Indicates that the counter has overflowed when set. This flag is a sticky flag that must be explicitly cleared by software.

15.10.3. IA32_PEBS_ENABLE MSR

In an IA-32 processor with Hyper-Threading Technology, PEBS is enabled and qualified with two bits in the IA32_PEBS_ENABLE MSR: bit 25 (ENABLE_PEBS_MY_THR) and 26 (ENABLE_PEBS_OTH_THR) respectively. These bits do not explicitly identify a specific logical processor by logic processor ID(T0 or T1); instead, they allow a software agent to enable PEBS for subsequent threads of execution on the same logical processor on which the agent is running ("my thread") or for the other logical processor in the physical package on which the agent is not running ("other thread").

PEBS is supported for only a subset of the at-retirement events: Execution_event, Front_end_event, and Replay_event. Also, PEBS can be carried out only with two performance counters: MSR_IQ_CCCR4 (MSR address 370H) for logical processor 0 and MSR_IQ_CCCR5 (MSR address 371H) for logical processor 1.

NOTE

Performance monitoring tools should use a processor affinity mask to bind the kernel mode components that need to modify the ENABLE_PEBS_MY_THR and ENABLE_PEBS_OTH_THR bits in the IA32_PEBS_ENABLE MSR to a specific logical processor to prevent these kernel mode components from migrating between different logical processors due to OS scheduling.

15.10.4. Performance Monitoring Events

All of the events listed in Table A-1 and A-2 are available in an Intel Xeon processor MP. When Hyper-Threading Technology is active, many performance monitoring events can be can be qualified by the logical processor ID, which corresponds to bit 0 of the initial APIC ID. This allows for counting an event in any or all of the logical processors. However, not all the events have this logic processor specificity, or thread specificity.

Here, each event falls into one of two categories:

- Thread specific (TS). The event can be qualified as occurring on a specific logical processor.
- Thread independent (TI). The event cannot be qualified as being associated with a specific logical processor.

Table A-6 gives logical processor specific information (TS or TI) for each of the events described in Tables A-1 and A-2.

If for example, a TS event occurred in logical processor T0, the counting of the event (as shown in Table 15-6) depends only on the setting of the T0_USR and T0_OS flags in the ESCR being used to set up the event counter. The T1_USR and T1_OS flags have no effect on the count.

Table 15-6. Effect of Logical Processor and CPL Qualification for Logical-Processor-Specific (TS) Events

	T1_OS/T1_USR=00	T1_OS/T1_USR=01	T1_OS/T1_USR=11	T1_OS/T1_USR=10
T0_OS/T0_USR=00	Zero count	Counts while T1 in USR	Counts while T1 in OS or USR	Counts while T1 in OS
T0_OS/T0_USR=01	Counts while T0 in USR	Counts while T0 in USR or T1 in USR	Counts while (a) T0 in USR or (b) T1 in OS or (c) T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS
T0_OS/T0_USR=11	Counts while T0 in OS or USR	Counts while (a) T0 in OS or (b) T0 in USR or (c) T1 in USR	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) or T0 in USR or (c) T1 in OS
T0_OS/T0_USR=10	Counts T0 in OS	Counts T0 in OS or T1 in USR	Counts while (a)T0 in Os or (b) T1 in OS or (c) T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS

When a bit in the event mask field is TI, the effect of specifying bit-0-3 of the associated ESCR are described in Table 15-6. For events that are marked as TI in Appendix A, the effect of selectively specifying T0_USR, T0_OS, T1_USR, T1_OS bits is shown in Table 15-6.

Table 15-7. Effect of logical processor and CPL qualification for non-logical-processor-specific (TI) events

	T1_OS/T1_USR=00	T1_OS/T1_USR=01	T1_OS/T1_USR=11	T1_OS/T1_USR=10
T0_OS/T0_USR=00	Zero count	Counts while (a) T0 in USR or (b) T1 in USR	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T1 in OS
T0_OS/T0_USR=01	Counts while (a) T0 in USR or (b) T1 in USR	Counts while (a) T0 in USR or (b) T1 in USR	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1
T0_OS/T0_USR=11	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1
T0_OS/T0_USR=10	Counts while (a) T0 in OS or (b) T1 in OS	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T1 in OS

15.11. PERFORMANCE MONITORING (P6 FAMILY PROCESSOR)

The P6 family processors provide two 40-bit performance counters, allowing two types of events to be monitored simultaneously. These counters can either count events or measure duration. When counting events, a counter is incremented each time a specified event takes place or a specified number of events takes place. When measuring duration, a counter counts the number of processor clocks that occur while a specified condition is true. The counters can count events or measure durations that occur at any privilege level. Table A-7 in Appendix A, *Performance-Monitoring Events*, lists the events that can be counted with the P6 family performance monitoring counters.

NOTE

> The performance-monitoring event listed in Appendix A, *Performance-Monitoring Events* are intended to be used as guides for performance tuning. The counter values reported are not guaranteed to be absolutely accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

The performance-monitoring counters are supported by four MSRs: the performance event select MSRs (PerfEvtSel0 and PerfEvtSel1) and the performance counter MSRs (PerfCtr0 and PerfCtr1). These registers can be read from and written to using the RDMSR and WRMSR instructions, respectively. They can be accessed using these instructions only when operating at privilege level 0. The PerfCtr0 and PerfCtr1 MSRs can be read from any privilege level using the RDPMC (read performance-monitoring counters) instruction.

NOTE

> The PerfEvtSel0, PerfEvtSel1, PerfCtr0, and PerfCtr1 MSRs and the events listed in Table A-7 are model-specific for P6 family processors. They are not guaranteed to be available in future IA-32 processors.

15.11.1. PerfEvtSel0 and PerfEvtSel1 MSRs

The PerfEvtSel0 and PerfEvtSel1 MSRs control the operation of the performance-monitoring counters, with one register used to set up each counter. They specify the events to be counted, how they should be counted, and the privilege levels at which counting should take place. Figure 15-17 shows the flags and fields in these MSRs.

The functions of the flags and fields in the PerfEvtSel0 and PerfEvtSel1 MSRs are as follows:

Event select field (bits 0 through 7)
Selects the event to be monitored (see Table A-7, for a list of events and their 8-bit codes).

Unit mask (UMASK) field (bits 8 through 15)
Further qualifies the event selected in the event select field. For example, for some cache events, the mask is used as a MESI-protocol qualifier of cache states (see Table A-7).

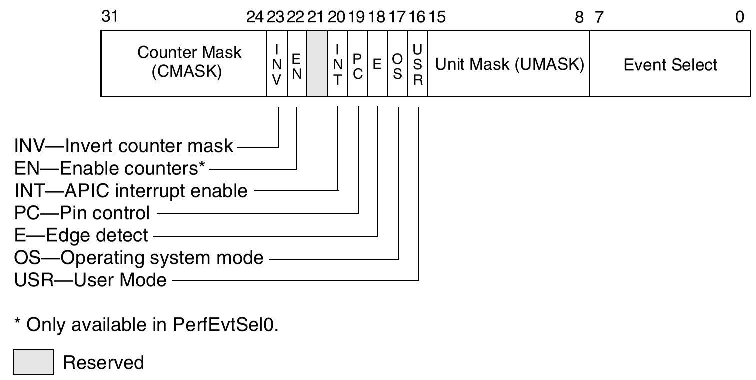

Figure 15-17. PerfEvtSel0 and PerfEvtSel1 MSRs

USR (user mode) flag (bit 16)
Specifies that events are counted only when the processor is operating at privilege levels 1, 2 or 3. This flag can be used in conjunction with the OS flag.

OS (operating system mode) flag (bit 17)
Specifies that events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USR flag.

E (edge detect) flag (bit 18)
Enables (when set) edge detection of events. The processor counts the number of deasserted to asserted transitions of any condition that can be expressed by the other fields. The mechanism is limited in that it does not permit back-to-back assertions to be distinguished. This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).

PC (pin control) flag (bit 19)
When set, the processor toggles the PM*i* pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PM*i* pins when the counter overflows. The toggling of a pin is defined as assertion of the pin for a single bus clock followed by deassertion.

INT (APIC interrupt enable) flag (bit 20)
When set, the processor generates an exception through its local APIC on counter overflow.

EN (Enable Counters) Flag (bit 22)
This flag is only present in the PerfEvtSel0 MSR. When set, performance counting is enabled in both performance-monitoring counters; when clear, both counters are disabled.

INV (invert) flag (bit 23)
Inverts the result of the counter-mask comparison when set, so that both greater than and less than comparisons can be made.

Counter mask (CMASK) field (bits 24 through 31)
When nonzero, the processor compares this mask to the number of events counted during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented. This mask can be used to count events only if multiple occurrences happen per clock (for example, two or more instructions retired per clock). If the counter-mask field is 0, then the counter is incremented each cycle by the number of events that occurred that cycle.

15.11.2. PerfCtr0 and PerfCtr1 MSRs

The performance-counter MSRs (PerfCtr0 and PerfCtr1) contain the event or duration counts for the selected events being counted. The RDPMC instruction can be used by programs or procedures running at any privilege level and in virtual-8086 mode to read these counters. The PCE flag in control register CR4 (bit 8) allows the use of this instruction to be restricted to only programs and procedures running at privilege level 0.

The RDPMC instruction is not serializing or ordered with other instructions. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDPMC instruction operation is performed.

Only the operating system, executing at privilege level 0, can directly manipulate the performance counters, using the RDMSR and WRMSR instructions. A secure operating system would clear the PCE flag during system initialization to disable direct user access to the performance-monitoring counters, but provide a user-accessible programming interface that emulates the RDPMC instruction.

The WRMSR instruction cannot arbitrarily write to the performance-monitoring counter MSRs (PerfCtr0 and PerfCtr1). Instead, the lower-order 32 bits of each MSR may be written with any value, and the high-order 8 bits are sign-extended according to the value of bit 31. This operation allows writing both positive and negative values to the performance counters.

15.11.3. Starting and Stopping the Performance-Monitoring Counters

The performance-monitoring counters are started by writing valid setup information in the PerfEvtSel0 and/or PerfEvtSel1 MSRs and setting the enable counters flag in the PerfEvtSel0 MSR. If the setup is valid, the counters begin counting following the execution of a WRMSR instruction that sets the enable counter flag. The counters can be stopped by clearing the enable counters flag or by clearing all the bits in the PerfEvtSel0 and PerfEvtSel1 MSRs. Counter 1 alone can be stopped by clearing the PerfEvtSel1 MSR.

15.11.4. Event and Time-Stamp Monitoring Software

To use the performance-monitoring counters and time-stamp counter, the operating system needs to provide an event-monitoring device driver. This driver should include procedures for handling the following operations:

- Feature checking.
- Initialize and start counters.
- Stop counters.
- Read the event counters.
- Read the time-stamp counter.

The event monitor feature determination procedure must determine whether the current processor supports the performance-monitoring counters and time-stamp counter. This procedure compares the family and model of the processor returned by the CPUID instruction with those of processors known to support performance monitoring. (The Pentium and P6 family processors support performance counters.) The procedure also checks the MSR and TSC flags returned to register EDX by the CPUID instruction to determine if the MSRs and the RDTSC instruction are supported.

The initialize and start counters procedure sets the PerfEvtSel0 and/or PerfEvtSel1 MSRs for the events to be counted and the method used to count them and initializes the counter MSRs (PerfCtr0 and PerfCtr1) to starting counts. The stop counters procedure stops the performance counters. (See Section 15.11.3., "Starting and Stopping the Performance-Monitoring Counters", for more information about starting and stopping the counters.)

The read counters procedure reads the values in the PerfCtr0 and PerfCtr1 MSRs, and a read time-stamp counter procedure reads the time-stamp counter. These procedures would be provided in lieu of enabling the RDTSC and RDPMC instructions that allow application code to read the counters.

15.11.5. Monitoring Counter Overflow

The P6 family processors provide the option of generating a local APIC interrupt when a performance-monitoring counter overflows. This mechanism is enabled by setting the interrupt enable flag in either the PerfEvtSel0 or the PerfEvtSel1 MSR. The primary use of this option is for statistical performance sampling.

To use this option, the operating system should do the following things on the processor for which performance events are required to be monitored:

- Provide an interrupt vector for handling the counter-overflow interrupt.
- Initialize the APIC PERF local vector entry to enable handling of performance-monitor counter overflow events.
- Provide an entry in the IDT that points to a stub exception handler that returns without executing any instructions.
- Provide an event monitor driver that provides the actual interrupt handler and modifies the reserved IDT entry to point to its interrupt routine.

When interrupted by a counter overflow, the interrupt handler needs to perform the following actions:

- Save the instruction pointer (EIP register), code-segment selector, TSS segment selector, counter values and other relevant information at the time of the interrupt.
- Reset the counter to its initial setting and return from the interrupt.

An event monitor application utility or another application program can read the information collected for analysis of the performance of the profiled application.

15.12. PERFORMANCE MONITORING (PENTIUM PROCESSORS)

The Pentium processor provides two 40-bit performance counters, which can be used either to count events or measure duration. The performance-monitoring counters are supported by three MSRs: the control and event select MSR (CESR) and the performance counter MSRs (CTR0 and CTR1). These registers can be read from and written to using the RDMSR and WRMSR instructions, respectively.

They can be accessed using these instructions only when operating at privilege level 0. Each counter has an associated external pin (PM0/BP0 and PM1/BP1), which can be used to indicate the state of the counter to external hardware.

NOTES

The CESR, CTR0, and CTR1 MSRs and the events listed in Table A-7 are model-specific for the Pentium processor.

The performance-monitoring event listed in Appendix B, *Model-Specific Registers (MSRs)* are intended to be used as guides for performance tuning. The counter values reported are not guaranteed to be absolutely accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

15.12.1. Control and Event Select Register (CESR)

The 32-bit control and event select MSR (CESR) is used to control the operation of performance-monitoring counters CTR0 and CTR1 and their associated pins (see Figure 15-18). To control each counter, the CESR register contains a 6-bit event select field (ES0 and ES1), a pin control flag (PC0 and PC1), and a 3-bit counter control field (CC0 and CC1). The functions of these fields are as follows:

ES0 and ES1 (event select) fields (bits 0 through 5, bits 16 through 21)
Selects (by entering an event code in the field) up to two events to be monitored. See Table A-7 for a list of available event codes

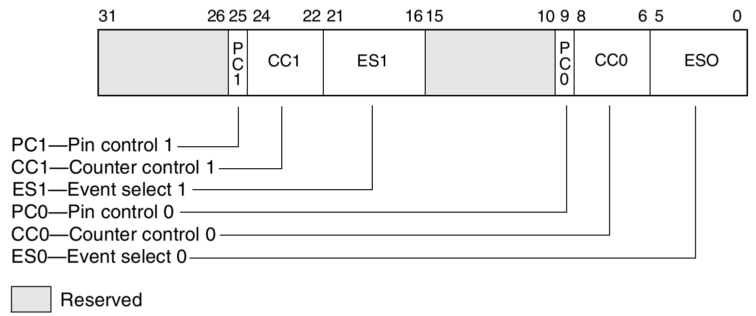

Figure 15-18. CESR MSR (Pentium Processor Only)

CC0 and CC1 (counter control) fields (bits 6 through 8, bits 22 through 24)
Controls the operation of the counter. The possible control codes are as follows:

CC*n*	**Meaning**
000	Count nothing (counter disabled)
001	Count the selected event while CPL is 0, 1, or 2
010	Count the selected event while CPL is 3
011	Count the selected event regardless of CPL
100	Count nothing (counter disabled)
101	Count clocks (duration) while CPL is 0, 1, or 2
110	Count clocks (duration) while CPL is 3
111	Count clocks (duration) regardless of CPL

Note that the highest order bit selects between counting events and counting clocks (duration); the middle bit enables counting when the CPL is 3; and the low-order bit enables counting when the CPL is 0, 1, or 2.

PC0 and PC1 (pin control) flags (bit 9, bits 25)
Selects the function of the external performance-monitoring counter pin (PM0/BP0 and PM1/BP1). Setting one of these flags to 1 causes the processor to assert its associated pin when the counter has overflowed; setting the flag to 0 causes the pin to be asserted when the counter has been incremented. These flags permit the pins to be individually programmed to indicate the overflow or incremented condition. Note that the external signalling of the event on the pins will lag the internal event by a few clocks as the signals are latched and buffered.

While a counter need not be stopped to sample its contents, it must be stopped and cleared or preset before switching to a new event. It is not possible to set one counter separately. If only one event needs to be changed, the CESR register must be read, the appropriate bits modified, and all bits must then be written back to CESR. At reset, all bits in the CESR register are cleared.

15.12.2. Use of the Performance-Monitoring Pins

When the performance-monitor pins PM0/BP0 and/or PM1/BP1 are configured to indicate when the performance-monitor counter has incremented and an “occurrence event” is being counted, the associated pin is asserted (high) each time the event occurs. When a “duration event” is being counted the associated PM pin is asserted for the entire duration of the event. When the performance-monitor pins are configured to indicate when the counter has overflowed, the associated PM pin is not asserted until the counter has overflowed.

When the PM0/BP0 and/or PM1/BP1 pins are configured to signal that a counter has incremented, it should be noted that although the counters may increment by 1 or 2 in a single clock, the pins can only indicate that the event occurred. Moreover, since the internal clock frequency may be higher than the external clock frequency, a single external clock may correspond to multiple internal clocks.

A “count up to” function may be provided when the event pin is programmed to signal an overflow of the counter. Because the counters are 40 bits, a carry out of bit 39 indicates an overflow. A counter may be preset to a specific value less then $2^{40} - 1$. After the counter has been enabled and the prescribed number of events has transpired, the counter will overflow.

Approximately 5 clocks later, the overflow is indicated externally and appropriate action, such as signaling an interrupt, may then be taken.

The PM0/BP0 and PM1/BP1 pins also serve to indicate breakpoint matches during in-circuit emulation, during which time the counter increment or overflow function of these pins is not available. After RESET, the PM0/BP0 and PM1/BP1 pins are configured for performance monitoring, however a hardware debugger may reconfigure these pins to indicate breakpoint matches.

15.12.3. Events Counted

The events that the performance-monitoring counters can set to count and record in the CTR0 and CTR1 MSRs are divided into two categories: occurrences and duration. Occurrences events are counted each time the event takes place. If the PM0/BP0 or PM1/BP1 pins are configured to indicate when a counter increments, they ar asserted each clock the counter increments. Note that if an event can happen twice in one clock, the counter increments by 2, however, the pins are asserted only once.

For duration events, the counter counts the total number of clocks that the condition is true. When configured to indicate when a counter increments, the PM0/BP0 and/or PM1/BP1 pins are asserted for the duration of the event.

Table A-7 lists the events that can be counted with the Pentium processor performance-monitoring counters.