Memory Ordering On Dynamic Execution (Pentium® Pro Family) Processors

Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications. Intel may make changes to specifications and product descriptions at any time, without notice.

Copyright (c) Intel Corporation 1997.

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

CONTENTS:

1.0. INTRODUCTION

The internal speculative nature of the Pentium Pro family and Pentium® II Architecture needs the ability to re-order certain memory operations, where allowed, to enable the extraction of the maximum processor performance. The Pentium Pro family and Pentium II processors provide the memory consistency model called speculative processor ordering.

Memory requests to the L2 cache or system memory go through the memory reorder buffer, which functions as a scheduling and dispatch station. This unit keeps track of all memory requests and is able to reorder some requests (*weakly ordered*) to prevent blocks and improve throughput. When processor ordering is violated, the Pentium Pro family and Pentium II implementation aborts all instructions under execution, beginning with the illegally completed load, and then resumes. The Pentium Pro family Pentium II processors have much deeper write buffer than the previous Intel486™ and Pentium processors. The write buffer is very dynamic, which has between 0 and 12 entries, whereas the Pentium processor has only two write buffers, one corresponding to each of the pipelines.

Memory ordering is predominantly a multiprocessor (MP) issue. Although a multiprocessor style, system environment can be constructed from a single CPU and I/O bus masters. Therefore, the uniprocessor (UP) memory accesses are the main concern in this document.

This document describes the memory ordering rules. A memory access A passes another memory access B means that B precedes A in the processor's Von Neumann execution stream, but that the processor executes A before B has completed, or even begun, because B and A are considered not to conflict. To simplify, the program flow is B then A, but the execution is A then B. The next section will illustrate in detail all the rules that govern the memory access ordering.

2.0. Memory Ordering Rules

2.1. Reads

- Reads are not perceived as passing reads (MP)

- Reads do not pass conflicting writes

```
mov cx, 0aaah
xor bx, bx
mov word ptr ds:[esi], cx
mov bx, word ptr
ds:[esi] ; This will not pass the previous conflicting write
cmp bx, 0aaah ; bx will contain 0aaah instead of 0
```

- Reads to speculatable memory types may occur randomly

Speculative memory types include Write Back (WB), Uncached Speculative Write Combined (USWC), Write Through (WT), and Write Protected (WP). For a detailed description of these memory types, please refer to the collateral titled "Information on Caches and Optimizing Memory Transfers".

This is a result of speculative execution in the P6 implementation. Software and systems should not depend on the details of when, and to what addresses, reads to these memory types occur for correctness. Particular implementations may generate these read prefetches or speculations at arbitrary times.

- Reads may not pass serializing instructions like CPUID

- Reads may not pass I/O instructions

- Reads may not pass locked instructions like XCHG

Due to the length of the pipeline, reads do pass locked instructions, but upon execution of the locked instruction, the entire pipeline is flushed and the instruction prefetech restarts after the locked instruction.

- Reads may pass Buffered Stores

2.2. Writes

- Writes do not occur randomly

Writes occur in the order of instruction execution. With the WB and UC memory types, the bus trace might not appear in order, but the writes committed to the cache are in order.

- Writes occur only for instructions that are actually executed

```
xor bx, bx ; clear the CF
mov bx, 5555h
mov cx, 0aaah
mov word ptr ds:[esi], cx
jnc next
mov word ptr ds:[esi], bx ;This is never executed
```

next: ;word ptr ds:[esi] still contains 0aaah

- Writes do not pass writes

Writes stored in the write buffer are always written to memory in program order.

- Writes may be buffered
- Writes do not pass I/O instructions
- Writes do not pass serializing instructions

For a description of the serializing instructions, please refer to section 2.4. Serializing Instructions.

- Writes do not pass synchronizing instructions

For a description of the synchronizing instructions, please refer to section 2.5. Locked Operations.

2.3. Instruction Fetch

- Instruction fetch does not pass conflicting writes

Self-modifying code is detected and signaled on the next instruction boundary. Signaling means that all pre-fetched instructions are flushed from the pipeline. The processor restarts prefetch beginning at the instruction following the write. Self-modifying code, therefore, exacts a great performance penalty.

```
mov ax, code_alias_seg
mov fs, ax
mov edi, offset foo
xor ebx, ebx
mov byte ptr fs:[edi], 90h ; Replace the code segment: code_alias_seg at foo
mov dword ptr fs:[edi+1], 90909090h ; with a NOP
foo:
mov ebx, 0fca50fh ; This will be overwritten by the last two moves.
cmp ebx, 0h
```

The register ebx should contain a 0h, since the code after "foo:" will not be executed until the 2 moves before "foo:" are done. Thus, the line "mov ebx, 0fca50fh" is overwritten by 5 NOPs.

- Instruction fetch may not pass instruction fetches

Remember, only the execution unit is dynamic and speculative.

- Instruction fetch may pass reads
- Reads may not pass instruction fetch
- Instruction fetch may pass I/O
- Instruction fetch may pass locked operations like XCHG
- Instruction fetch may not pass serializing instructions

Due to the length of the processor's instruction pipeline, instruction fetches do often pass serializing instructions. However, upon execution of the serializing instruction, the entire pipeline is flushed and the instruction prefetch restarts after the serializing instruction.

2.4. Serializing Instructions

Serializing instructions constrain speculative execution. Therefore, when serializing instructions are executed, the dynamic execution feature of the Pentium Pro Family and Pentium II is defeated. The following instructions are serializing:

Move to special register (include WRMSR);

INVD, INVPG, WBINVD;

IRET, IRETD, LGDT, LLDT, LIDT, LTR;

CPUID;

RSM;

- Instruction fetch does not pass serializing instructions
- Serializing instructions do not pass anything

They wait for all previous instructions to complete, and for all write instructions buffered by the CPU to drain.

- Nothing passes a serializing instruction

Instruction prefetch may have occurred before a serializing instruction, but actual instruction fetch and execution does not occur; even if it does occur, a re-fetch is necessary.

2.5. Locked Operations

Locked operations include XCHG, CMXCHG, CMPXCHG8B, and read-modify-write instructions to which the LOCK prefix can be applied.

- Locked operations do not pass anything

Locked operations wait for all previous instructions to complete. They wait for on-chip buffers to drain, and for external buffers to be emptied..

- Instruction fetch may pass locked operations, but no other operations may pass

Locked operations synchronize data, but not instruction fetch or TLB (Translation Lookaside Buffer) miss handling. Data may be present in a cache or TLB, due to the speculative cacheability feature, which means that a lock cannot be used to prevent data from being fetched into a cache or TLB.

- Locked operations are atomic with respect to all other memory operations and all externally observable events

This means that no other operation can occur between the Read and Write of a locked RMW operation.

2.6. I/O Operations

- both IN and OUT instructions do not pass anything

Both type of instructions wait for all previous instructions to complete, for all writes buffered by the CPU to drain and for all previous stores to be globally observed.

- OUT instructions are not buffered internally

- Instruction fetch and page table walking may pass IN and OUT instructions, but nothing else may

Subsequent instructions, other than instruction fetch and page table walking, do not begin execution until the IN or OUT instruction has completed.

Instruction fetches (loads) and page table walks will pass I/O operations on the bus. But, the speculative snooping mechanism will ensure that they are re-performed, if a coherency side effect of the I/O is produced before the I/O cycle is completed.

2.7. Page Table Walking Accesses

The memory ordering model for page table walks is more relaxed in the Pentium Pro family and Pentium II processors than in previous processors.

- Page table walks can occur at any time, randomly

Page table walking to satisfy TLB (Translation Lookaside Buffer) misses can be performed speculatively and out-of-order; page table walks are subject to speculative cacheability.

- Entries may be placed into the TLB, at any time, randomly, as long as the translation is marked present in the PDEs and PTEs and paging is enabled

- Entries may be displaced from the TLB at any time, randomly

- Entries will not be placed in the TLB if marked non-present in the PDE (Page Description Entries) or PTEs (Page Table Entries), nor if paging is disabled

- Page table walk loads can pass loads, stores, locked instructions and serializing instructions

- Page table walk loads can be passed by loads

- Setting of the accessed and dirty bits is strongly ordered

The instructions are *strongly ordered* when instructions are executed in program order, resulting in accesses being issued in the order implied by the program. The setting of the accessed and dirty is treated the same as locked atomic RMW synchronization instruction.

3.0. CONCLUSION

It is recommended that software written to run on the Pentium Pro Family and Pentium II processors assume the processor-order model or a weaker memory-ordering model.

Despite the fact that processor ordering is supported by the Pentium Pro family and Pentium II

processors and previous CPUs, new code developments should employ locked atomic RMW (Read-Modify-Write) instructions like XCHG for synchronization, instead of relying on memory access ordering. Note that any of these strongly ordered instructions will cause a performance hit.

In a single-processor system for memory regions defined as write-back cacheable, the general rule of thumb is:

1. Reads can be carried out speculatively and in any order.
2. Reads can pass buffered writes, but the processor guarantees program correctness if the write is to the same memory location as the read.
3. Writes to memory are always carried out in program order.
4. Writes can be buffered.
5. Writes are not performed speculatively; they are only performed for instructions that have actually been executed.
6. Data writes can be forwarded within the processor.
7. Reads or writes cannot pass (be carried out ahead of) I/O instructions, locked instructions, or serializing instructions.
