IA-32 architecture coherency

serializing instructions and events				
doc? #1	instruction or event	486? #2	use? #3	description and comments
yes	IRET(D)	yes	yes	may be privileged under some circumstances
yes	RSM	yes	yes	can only be executed from within SMM
yes	CPUID	no	no	non-privileged
yes	LGDT Ms	no	no	privileged
yes	LIDT Ms	no	no	privileged
yes	LLDT Ew	no	no	privileged
yes	LTR Ew	no	no	privileged
yes	INVLPG M	no	no	privileged, implemented badly in Intel P5 and P54
yes	INVD	no	no	privileged, does not write back cache contents
yes	WBINVD	yes	yes	privileged
no	LMSW Ew	yes	no	privileged
yes	MOV CR0,Rd	yes	yes	privileged
yes	MOV CR2,Rd	no	no	privileged
yes	MOV CR3,Rd	no	no	privileged
yes	MOV CR4,Rd	no	no	privileged
yes	MOV DR0..7,Rd	yes	yes	privileged
yes	WRMSR	n/a	yes	privileged
yes	SFENCE	n/a	yes	non-privileged, but requires SSE
yes	MFENCE	n/a	yes	non-privileged, but requires SSE2
yes	LFENCE	n/a	yes	non-privileged, but requires SSE2
yes	RDTSCP	n/a	no	privileged if CR4.TSD=1, and requires TSCP
no	exceptions #4	yes	no	incl. INT Ib, INT1, INT3, INTO (taken), BOUND (taken)
no	interrupts #4	yes	no	INTR, NMI, SMI, INIT
no	branches	yes	no	CALL Ap/Ep/Ev/Jv, RET, RET Iw, RETF, RETF Iw JMP Ap/Ep/Ev/Jv/Jb, Jcc Jb/Jv (taken), JCXZ LOOP, LOOPE, LOOPNE
no	segment loads	no	no	LDS/LES/LFS/LGS/LSS Gv,MP POP DS/ES/FS/GS/SS MOV Sw,Ew
no	A20M# changes #5	yes	no	KBC or PS/2

notes	description
#1	Only the documented instructions and events are guaranteed to be serializing on future IA-32 processors.
#2	Serializing instructions and events were defined and documented starting with Intel's P5-core processors.
#3	To ensure backward compatibility it is not recommended to use these. (This depends on #1 and #2.)
#4	The nature of the IA-32 architecture implies that these instructions and events are serializing.
#5	In case of an OUTS instruction serialization isn't guaranteed until all iterations have been completed.

TLB invalidation	PDPTE-to-PDPTR reloading
• writes to CR3 #1 • changes to CR3 during a task switch #1 • changes to CR0.PE • changes to CR0.PG #2 • changes to CR4.PSE (if PSE is supported) #2 • changes to CR4.PGE (if PGE is supported)	• writes to CR3 #1 • changes to CR3 during a task switch #1, #2 • a 0-to-1 change of CR0.PG while CR4.PAE=1 #3 • a 0-to-1 change of CR4.PAE while CR0.PG=1 #3 • changes to CR4.PSE (if PSE is supported) #4 • changes to CR4.PGE (if PGE is supported) #4

- changes to CR4.PAE (if PAE is supported)
- INVLPG M instruction
- RSM instruction
- writes to MTRRs (if MTRRs are supported)
- writes to PAT MSR (if PAT is supported)
- writes to APIC_BASE MSR (if APIC is supported)
- SMI [#3]
- A20M# changes [#4]

notes	description
#1	global entries remain if PGE is supported
#2	not on Intel P5-core processors
#3	if TLB is used to implement SMM remapping
#4	if TLB is used to implement A20M#

- RSM instruction [#5]

notes	description
#1	while CR0.PG=1 and CR4.PAE=1
#2	Intel P4-core processors always reload
#3	a 1-to-0 change should set the PDPTRs to zero
#4	unnecessary, but done by Intel processors
#5	SMI should save the PDPTRs in the SSM, and then set them to zero (P6 doesn't, but P4 does)

---

store buffer draining

- processor exceptions and external interrupts
- serializing instructions (see above)
- I/O instructions (IN, (REP) INS, OUT, (REP) OUTS)
- LOCKed operations (explicit and implicit)
- SFENCE instruction (if SSE is supported)
- MFENCE instruction (if SSE2 is supported)
- reads from memory regions that are marked UC

MTRR conflicts

	UC	WC	WT	WP	WB
UC	**UC**	UC	UC	UC	UC
WC	UC	**WC**	UC	WC	UC
WT	UC	UC	**WT**	WT	WT
WP	UC	WC	WT	**WP**	WT
WB	UC	UC	WT	WT	**WB**

note	Because the behavior of the gray cases is reserved, it should not be relied upon. In essence the processor computes the logical AND of all the involved memory types, as shown in this table.

MTRR-PAT conflicts

		PAT					
		UC	WC	WT	WP	WB	UC-
MTRRs	UC	**UC**_M	#1	UC_M	UC_M	UC_M	UC_M
	WC	UC_P	**WC**	UC	UC	WC	**WC**
	WT	UC_P	WC	**WT**	#2	WT	UC_P
	WP	UC_P	WC	#2	**WP**	WP	UC_P
	WB	UC_P	WC	WT	WP	**WB**	UC_P

notes	description
#1	From an architectural standpoint the processor should honour MTRR_DEF_TYPE.E. While set to 0 the MTRRs are disabled, memory should be treated as UC, and PAT=WC should not be able to take precedence; thus the result should be UC_M. However, while set to 1 the MTRRs are enabled, and PAT=WC should be able to take precedence; thus the result should be WC. While Intel processors do honour the E bit, AMD processors do not -- for them PAT=WC always takes predence; thus their result is always WC.
#2	Because the behavior of this particular case is reserved, it shouldn't be relied upon. While Intel processors compute the logical AND, resulting in WT, AMD processors treat this combination as explicitly illegal, resulting in UC.