IA-32 architecture interrupts

external interrupts				
name	priority [#1]	sensitivity	bloackable?	latchable?
RESET	1	level [#3]	no	no
STPCLK	2 [#2]	level [#4]	no	no
SMI	3	edge	BLOCK_SMI [#6]	LATCH_SMI [#7]
INIT	4	edge	BLOCK_INIT [#6]	LATCH_INIT [#7]
NMI	5	edge	BLOCK_NMI [#6, #8]	LATCH_NMI [#7]
INTR	6	level [#5]	EFLAGS.IF=0	no

notes	description
#1	Unless noted otherwise the CPU will recognize assertions as short as 1 BCLK. However, the external interrupts(s) must remain asserted until microcode has read the processor's event status register. Note that this window may cause two or more physically consecutive events to appear as logically simultaneous; this may result in a high priority interrupt being serviced before an earlier lower priority interrupt. The size of this window is implementation-specific.
#2	Several South Bridges are sensitive with regard to the STOPGRANT latency. Thus STPCLK has higher priority than e.g. SMI, NMI, or INTR. (Older CPUs implement it below INTR though.)
#3	To ensure proper recognition, RESET must remain asserted for a CPU-specific duration.
#4	To ensure proper recognition, STPCLK must remain asserted until STOPGRANT was seen.
#5	To ensure proper recognition, INTR must remain asserted until the first INTACK was seen.
#6	These flags are affected by various events. Refer to MSRs, SMM, and initial state for details.
#7	One occurence of a signal can be latched while it is blocked, as long as BCLK is asserted.
#8	When the processor services a pending NMI, it first blocks further NMIs, and then attempts to enter the NMI handler. Because the latter may cause an exception, these two steps are not atomic: an exception handler could execute an IRET instruction, which would unblock NMIs. As a result, a subsequent NMI may interrupt the NMI handler. It is the user's responsibility to cope with this scenario. By contrast, if the processor services a pending INTR via an interrupt gate, it first performs all necessary checks (and causes the corresponding exceptions), and then clears FLAGS.IF.

external interrupt suppression			
SUPPRESS_INTERRUPTS flags [#1,#2]			
name	STI	POP SS	MOV SS,Ew
RESET	no	no	no
STPCLK [#3]	no	no	no
INIT	yes [#4]	yes [#4]	yes [#4]
SMI	yes [#5]	yes	yes
NMI	yes [#5]	yes	yes
INTR	yes	yes	yes
STPCLK [#3]	yes	yes	yes

notes	description
#1	A suppression of at least one instruction is guaranteed, though it may be longer. The suppression expires simultaneously for all the affected interrupts, to ensure their proper prioritization.
#2	If the suppressing instruction is followed by a HLT, by a REP string instruction, or by a FP instruction that causes a FP freeze, then the suppression will end eventually, to permit interrupt service. Also, for

	streams of suppressing instructions only the first suppression will be guaranteed. Last but not least, if the suppressing instruction is followed by a trap or a fault, then the suppression will end, so that the first instruction of the exception handler can be a suppressing one. Though the (#DB) trap could be suppressed to avoid this scenario, faults could not be. This means that the concept of external interrupt suppresion is flawed, and therefore should go away in future x86 processors. To properly handle all corner cases, the SUPPRESS_INTERRUPTS flag requires a 2-bit implementation.
#3	Processors that implement STPCLK "between" RESET and SMI do not suppress it, while processors that implement it "below" INTR do.
#4	INIT must be suppressed to ensure its proper prioritization versus SMI.
#5	Intel processors don't suppress SMI or NMI after an STI instruction. Since the INTR suppresion is not preserved across an SMI or NMI handler, this may result in an INTR being serviced after the STI, which constitutes a violation of the INTR suppresion. Therefore, ideally the STI instruction also suppresses SMI and NMI.