IA-32 architecture
legacy stuff

KBC, PS/2, and A20M#

KBC

output port, pin 1			
0	A20M# low	asserted	wrap memory
1	A20M# high	deasserted	flat memory

PS/2

system Port A, bit 1			
0	KBC's A20M#	depends	depends
1	A20M# high	deasserted	flat memory

A20M#

KBC			PS/2			A20M#	
L	0	wrap	L	0	pass	L	wrap [#1]
H	1	flat	L	0	pass	H	flat [#2]
L	0	wrap	H	1	flat	H	flat [#3]
H	1	flat	H	1	flat	H	flat [#3]

note	description
#1	Ideally the processor should honour A20M# even when it is not in real mode, but it should ignore A20M# when it is in SMM. However, some processors don't implement it that way.
#2	This is the default state after RESET, which gives (historical) precedence to the KBC.
#3	The PS/2 state after INIT is chipset-specific. Older chipsets (eg. PIIX) leave it unchanged. Newer chipsets (eg. ICH) set it to flat.

FERR# and IGNNE#

For legacy purposes, today's x86 processors support the FERR# output signal and the IGNNE# input signal. In general the FERR# signal represents the state of the FSW.ES bit, while FSW.B is a copy of that bit. If the bit is 0, then FERR# is deasserted. If the bit is 1, then FERR# is asserted. Both FERR# and IGNNE# are level-sensitive; hence a processor may implement them via physical pins or via virtual packets.

FERR# assertion

Fundamentally the processor checks the mask bits in FCW and the signal bits in FSW: if an unmasked FPU exception is signaled, then FSW.B/ES gets set to 1, and FERR# gets asserted (unless it already is, of course). Said check may be performed right after an instruction that caused an unmasked FPU exception (this is known as immediate reporting) or it may be performed whenever a FPU or MMX instruction, or a MMX-SSE, MMX-SSE2, SSE-MMX or SSE2-MMX instruction is about to be executed (this is known as deferred reporting). Since FXSAVE and FXRSTOR don't perform this check, they don't assert FERR#.

Though a processor may implement either immediate or deferred reporting, or a combination thereof, it should always implement deferred reporting, because doing so allows to predict when the corresponding INTR is going to occur.

The assertion of FERR# is independent of the current state of CR0.NE or IGNNE#, and, in case of deferred reporting, it occurs at the beginning of the reporting instruction, just after the #UD and #NM conditions have been checked.

While CR0.NE is 0, the processor is in what is called "MS-DOS compatibility mode"; here the South Bridge is expected to generate an IRQ13 in response to the FERR# assertion, to signal the FPU exception back to the processor.

While CR0.NE is 1, the processor is in what is called "native mode"; here the #MF exception is used to signal the FPU exception; software is expected to either disable the IRQ13 generation in the South Bridge, or mask off IRQ13 via the interrupt controller, or ignore any INTRs caused by IRQ13. Waiting FPU instructions and MMX instructions do generate the #MF exception; no-wait FPU instructions, FXSAVE, and FXRSTOR don't (i.e. execution continues).

If FERR# is asserted by a FNCLEX, FNINIT, FNSAVE, or FNSTENV instruction, then the signal is deasserted

before the next instruction is executed, because these FPU instructions set FSW.B/ES to 0, and they do not wait. As a result the FERR# signal is pulsed only briefly in this case.

If FERR# is asserted by any other no-wait FPU instruction (read: FNENI, FNDISI, FNSETPM, FNSTCW, FNSTSW, or FNSTSW AX), waiting FPU instruction (FWAIT and all others), MMX instruction (EMMS and all others), or MMX-SSE, MMX-SSE2, SSE-MMX, or SSE2-MMX instruction, then it remains asserted until FSW.B/ES gets set to 0 later.

Note that FPU exception reporting should also be deferred if a FLDCW, FRSTOR, FLDENV, or FXRSTOR instruction is used to load a FPU state that includes an unmasked signaled FPU exception.

FERR# deassertion

As mentioned before, the processor always deasserts FERR# when FSW.B/ES gets set to 0.

During RESET the FPU is initialized; FSW.B/ES gets set to 0, and thus FERR# gets deasserted.
During INIT the FPU remains unmodified, and thus the state of FERR# remains unchanged.

Since the F(N)CLEX, F(N)INIT, F(N)SAVE, and F(N)STENV instructions clear FSW.B/ES, they also deassert FERR#. (The waiting versions can only be executed if IGNNE# is asserted -- otherwise the processor will freeze.)
By contrast, the FXSAVE instruction does not clear FSW.B/ES, and hence it does not deassert FERR#.

The FRSTOR, FLDENV, and FXRSTOR instructions should always deassert FERR#, regardless of whether the FPU state they load includes an unmasked signaled FPU exception or not. (As mentioned earlier, the reporting of any such exception should be deferred.)

FLDCW instructions deassert FERR# if they mask all signaled FPU exceptions; otherwise FERR# remains asserted.

IGNNE# behavior

If the processor is in "native mode" (CR0.NE is set to 1), then IGNNE# has no effect.

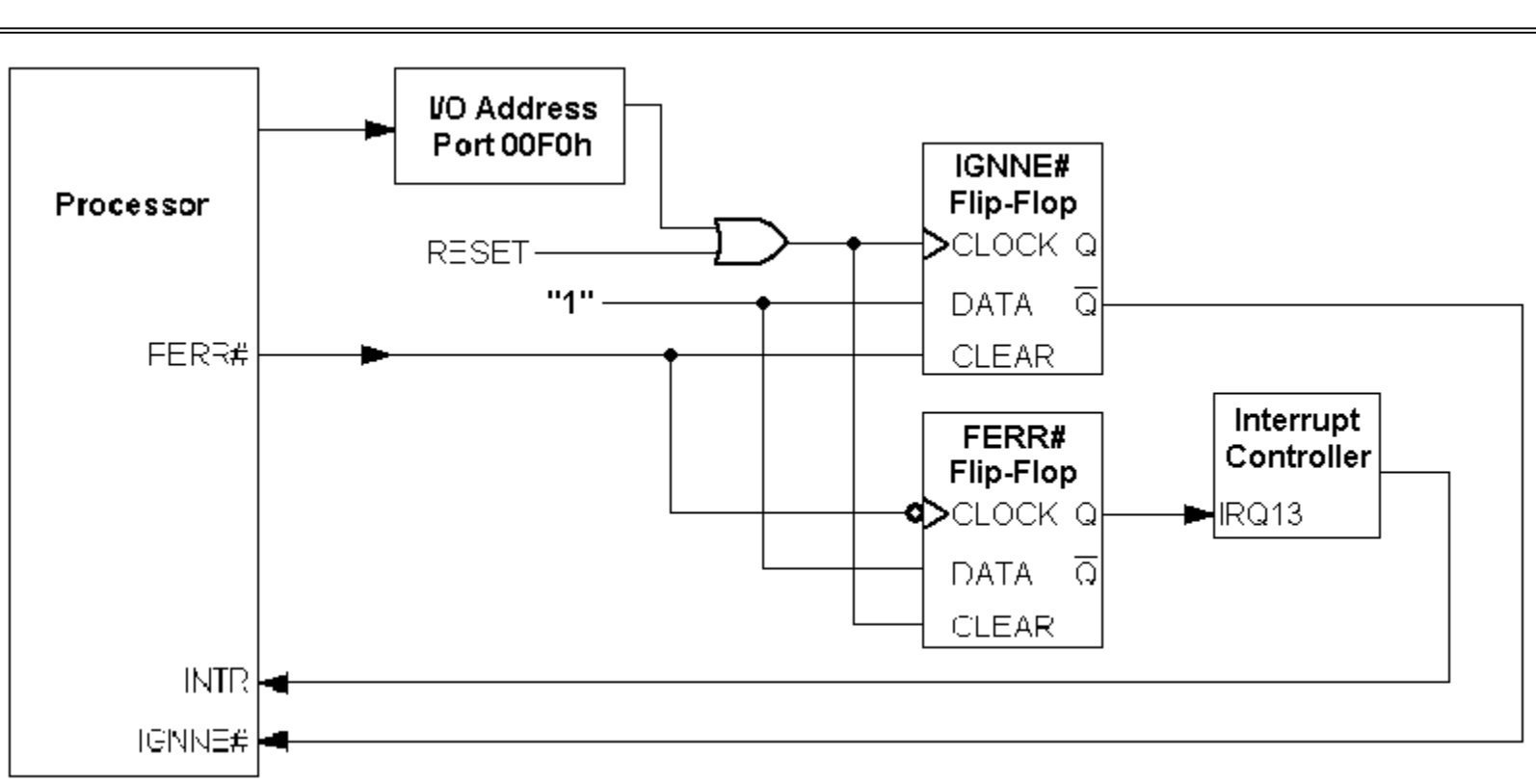
If the processor is in "MS-DOS compatibility mode" (CR0.NE is set to 0), and FERR# gets asserted by a no-wait FPU instruction, then execution will continue. However, if FERR# gets asserted by a waiting FPU instruction, or by a MMX instruction, or by a MMX-SSE, MMX-SSE2, SSE-MMX, or SSE2-MMX instruction, then the processor will freeze, to wait for an interrupt (i.e. RESET, INIT, SMI, NMI, or INTR [e.g. IRQ13]), or for the assertion of IGNNE#.

In essence asserting IGNNE# allows the processor to execute waiting FPU instructions, MMX instructions, MMX-SSE, MMX-SSE2, SSE-MMX, or SSE2-MMX instructions while an unmasked FPU exception is pending. This may be useful inside FPU exception handlers.

The processor should not (but might) implement the "interrupt sampling window" for no-wait FPU instructions, which is described in IASWDG volume 1, section D.2.1.3.

Last but not least, the processor is subject to the limitations described in IASWDG volume 1, section D.3.5.

FERR#, IGNNE#, IRQ13, and INTR



external logic for supporting FPU exceptions

FERR# **[1]** **[5]**

IGNNE#		**[4]**	**[5]**	
FP_IRQ	**[2]**	**[4]**		
IRQ2	**[2]**			**[6]**
IRQ13	**[2]**			**[6]**
INTR	**[3]**			

notes	description
#1	If an unmasked signaled FPU exception is detected, then the processor asserts FERR#.
#2	By default (i.e. this is optional) the chipset generates an IRQ13 in response to the assertion of FERR#. Depending on the actual implementation this may require some time, ie. cause a brief delay.
#3	The chipset asserts INTR, communicates vector PIC2_base+(13-8) to the processor (default: 75h), and then deasserts INTR. The processor enters the corresponding interrupt handler, specified by the IVT/IDT.
#4	The interrupt handler writes 00h to port F0h. In response the chipset asserts IGNNE#, allowing an interrupt handler to also execute waiting FPU instructions. In addition the chipset clears the FP_IRQ input to PIC2. Both, IRQ13 and IRQ2 are however still latched in the PICs. They will be handled in step #6 below.
#5	The interrupt handler clears FSW.B/ES. This causes the deassertion of FERR#. In response the chipset deasserts IGNNE#. This will prevent any further unmasked signaled FPU exceptions from being ignored.
#6	The interrupt handler issues an EOI to both PICs, to get rid of the IRQ13 and the IRQ2, and then returns.