# Interrupt Latency in 80386EX Based System

This document will attempt to explain the Interrupt latency and Interrupt Response Time for a 386EX based systems in Real Mode of operation. Similar explanations can be used with some modifications for the task switch and privilege level verifications, for the Protected Mode of operation, but however, this will be covered at a later time. Intel 80386EX CPU core is same as the 386SX core and hence most of the Interrupt latency descriptions for the SX core would apply directly.

As a start, some of the terminology which will be used in the course of this document will be explained (may be slightly different from other documentation).

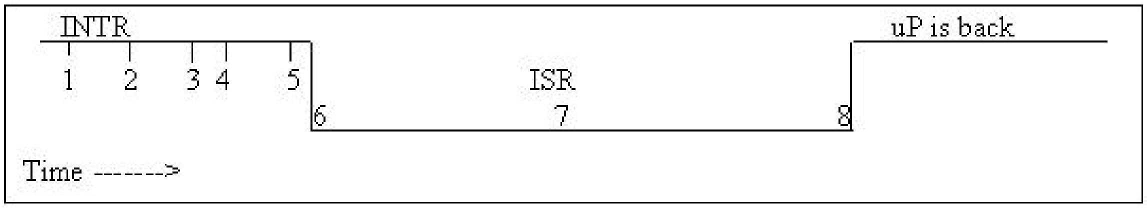
**Interrupt Latency**: The time that elapses before an interrupt request is serviced by the CPU (recognition of the interrupt by the CPU with an interrupt acknowledge cycle). Also called latency time.

**Interrupt response time**: Time that elapses between the occurrence of an interrupt and the execution of the first instruction of that Interrupt Service Routine (ISR) by the CPU.

In response to an interrupt, the following operations takes place in a microprocessor system while in the Real Mode of operation as indicated below. However, a very similar procedure would be followed in the protected mode except that the Task switch may be necessary and the privilege levels may cause the instruction executions to take longer to execute than in normal real mode operation [1].

- 1. Recognition of the INTR by the CPU.
- 2. The processing of the current instruction is completed.
- 3. Micro-code of the CPU Core executes the INTR (1 cycle for 386SX).
- 4. Get INTR vector from the 8259.
- 5. Branching to the ISR+ save the microprocessor state.
- 6. First instruction of the ISR executed.
- 7. Continue with interrupt service routine.
- 8. Restore the saved status of the microprocessor and return to the instruction that follows the interrupted instruction.

The following diagram illustrates the above sequence:



In embedded applications, usually the interrupt latency and interrupt response times are very important. With the above definitions, Latency time can be sum of the clock cycles from states 1 through 4. The Interrupt Response Time would be the sum of states 1 through 6. Following assumptions are made [2]:

- 1. Interrupts are not masked (applies to all maskable INTRs).
- 2. Only one NMI is encountered by the CPU at a time.
- 3. The LOCK bus activity or a long DMA transfer is not taking place at the time of INTR.

With this in mind, the calculations of the number of clock cycles taken for the above steps would be (these are only the simulated values and the actual values in a system may be different):

- 1. Recognition of the INTR: The 386SX core takes 8 CLK2 cycles or 4 CPU core clocks (c/s) to recognize an INTR.
- 2. Longest instruction in the Real Mode would be the IDIV (signed, double word) would be 48 c/s, assuming 0Wait state code memory, Wc=0 or 48+2*Wc. Here we assume that the instruction was pre-fetched in to the processor queue (in about 85% of cases this is true) and that the instruction had just started.
- 3. Micro-code of the CPU Core executes the INTR (1 cycle for 386SX).
- 4. Get INTR Vector from the 8259. 386EX has TWO 8259's in cascaded fashion. This would require 4+4+2 or 10 c/s (assuming zero wait states for the internal peripheral access, and would increase for the external 8259's depending upon the wait states).
- 5. Branching to the ISR and saving the uP state: Would be similar to INT *imm8* and would take 37 c/s in real mode, again assuming the SRAM memory is 0Wait state , Ws=0, or 37+3*Ws+5*Wc (Wc=Code memory wait state).
- 6. First Instruction of the ISR being executed: Assuming the pre-fetch queue is full, this should be equal to 0 c/s, if pre-fetch queue is not full, it would take additional 3 c/s.
- 7. Interrupt Service routine: A simple NULL INTR (For USR case) is shown below with number of clock cycles it would take to execute, assuming a 0Wait state memory (Wc=0), if the code memory has wait state then that should be added appropriately.

```
Instructions                                       0 Wait 1 Wait

ISR0: push ax   ; save the accumulator                    2 c/s   3 c/s
push dx         ; save dx register contents               2 c/s   3 c/s
mov dx, s0sts   ; get the Serial status reg               2 c/s   3 c/s
in ax, dx       ; status word in                          12 c/s  13 c/s
Test ax, imm16 ; check for the Transmit                   2 c/s   3 c/s
Jcc Label       ; if not this is RxInt                    7/3 c/s 8/4 c/s
pop dx                                                    2 c/s   3 c/s
pop ax                                                    2 c/s   3 c/s
IRET            ; Just execute return for TxInt           22 c/s  24 c/s
```

Total of 2+2+2+12+2+3+2+2+22 = 49 c/s for branch not taken, in the above case, with 0Wait state (Wc=0) or 49+9*Wc+5*Ws.

8. The processor is back to what it was doing before the INTR occurred. This should again take zero c/s, if the pre fetch queue is full else add 3 c/s for the pre-fetch queue to be filled.

So the total clock cycles would be, for the NULL INTR handling in this case is = 4+48+1+10+37+0+49+0=149 c/s for a ZERO WAIT STATE system (or 4+ 48+2*Wc+1+10+37+3*Ws+5*Wc+0+49+9*Wc+5*Ws c/s or 149+16*Wc+8*Ws c/s for systems with Wait states, again, assuming that the pre-fetch queue is full).

*Interrupt Latency or Latency Time=4+48+1+10=63c/s with 0Wait states.*

*Interrupt Response Time= 4+48+1+10+37+0=100c/s with 0Wait states.*

*Time take to handle the NULL INTR is 49 c/s**

*Time taken for states 3- 8=49+37+10+1=87c/s* ( overhead due to NULL INTR).*

- *these times are simulated numbers and may be lesser in an actual system.*

**References:**

1. K.L Short, *Microprocessors and Programmed Logic*, Prentice Hill, Inc, Englewood Cliffs, NJ 07632, Second Edition, pp 320.

2. *386SX Microprocessor, Hardware Reference Manual*, Intel, 1990, pp. 3-16, 3-28.