https://github.com/justinnsyy/SYSC4001_A1

The interrupt process includes many steps, the main one being the handling of the interrupt, which is done by the Interrupt Service Routine (ISR). But before the ISR is executed, the Operating System (OS) must first complete the preparation steps. These include, saving the context, switching from user to kernel mode, calculating where in memory the ISR start address is based on the interrupt number, and getting the ISR address from the vector table.

These are overall negligible in terms of time taken (~1ms), but increasing the save/restore context time from 10, to 20, to 30ms, creates an increase in the overhead time for the Interrupt Service Routine (ISR), thus increasing the interrupt latency. This increase is linear with the amount of time it takes to perform the context switch, meaning doubling the context switch doubles that portion of interrupt overhead. This can negatively impact the system as the interrupts take longer to execute, resulting in delays in completing user tasks. It is also possible for a backlog of interrupts to build up, as the device may generate interrupts faster than the CPU can handle them, as the context switch overhead is high. While these are important issues to address, most importantly, this wastes the CPU, as it wastes cycles and spends more time performing context switches than meaningful calculations, reducing throughput overall.

Now moving on to the second part of the interrupt process, which is the execution of the ISR, and how varying its activity time between 40 and 200ms can also produce negative results. As the ISR activity time increases, the total runtime increases too, in a nearly linear fashion similar to increasing the context switch time. Throughput is reduced here too as the CPU uses more cycles to execute the body of the ISR, especially if the activity time begins to approach the 200ms mark. Again, the user tasks also begin to become delayed, along with other interrupt driven events, as the interrupt processing time begins to greatly increase.

> When considering each interrupt, total overhead is equated by the sum of context, vector fetch, IRET, and ISR body. Because IRET costs are tiny compared to the ISR, changes to these micro-steps have smaller linear impact, while the ISR and context times dominate.
>
> Assuming we keep the fetch/loopup steps at 1 ms each, if we increase the address from 2 bytes to 4 bytes, there would not be a drastic change unless you alter the steps to take more time. For instance, if “finding the vector” and “getting the ISR address” takes 1 ms slower, the total run time would grow by (extra time per step) x (# of interrupts).
>
> A faster CPU mainly shortens the CPU burst lines in your trace, but it doesn’t speed up device delays or the interrupt steps. So the benefit depends on how much time you spent doing CPU bursts before. If CPU bursts were 40% of the run and you make the CPU 2× faster, the total time becomes about 60% (unchanged) + 20% = 80% of before—only a 1.25× speedup. If your run is I/O-bound (lots of waiting on devices),

speeding up the CPU helps only a little; reducing ISR time or the number of interrupts helps more