Building upon Assignment 1, this lab goal was to expand the scope of the interrupt simulation to analyze the effects of different context-switch times and ISR durations on overall execution performance, while still working with the original device and vector tables.

The simulator was designed to execute all the steps of the interrupt sequence, including entry into and exit from the kernel, context saving and restoration, saving and restoration of the CPU state, ISR address lookup, execution of the ISR, and checking and transferring the data from the device to the memory.

Output trace files were created and stored for the different configurations, which were:

Context = 10 ms, ISR = 40 ms

Context = 10 ms, ISR = 80 ms

Context = 10 ms, ISR = 120 ms

Context = 10 ms, ISR = 160 ms

Context = 20 ms, ISR = 40 ms

Results and Analysis

As in A1, context save/restore time still has a direct proportional impact on total runtime. The overhead caused by incremental context switch commands during execution and return, will (most likely) explain the additional 10-12% increase in overall execution time every time the context time was increased from 10 to 20 ms.

The duration of the ISRs remained the most significant contributor to the total execution time. Specifically, having the ISRs last 40ms to 160ms made the elapsed time quadruple, which shows that the device service routines determine the latency of the CPU and the starvation of the processes. After 120ms, the kernel spent most of its time servicing interrupts, so there was less time available to return to user mode, reducing overall system throughput and increasing the time other processes had to wait before they could start execution, thus increasing response delays.

Execution logs exhibited identical operation patterns, each having alternating CPU bursts of 100 ms and 50 ms interspersed with SYSCALL and END_IO activities. Logs also grew in line numbers corresponding to context and ISR parameters, indicating a rise in kernel activities.

This confirms that interrupt latency leaves a ceiling on system performance. Improvements in context save time and interrupt service routines increase system efficiency. While the use of a faster CPU can help reduce the workload that interrupts have, software overhead in the kernel mode is still the biggest bottleneck. Having to make the adjustments in Assignment 2 helped to

prove that the demands of timing in real-time and embedded systems make the need to optimize for routing far more crucial.