Homework 2

Question 1 : Difference between polling-based input and interrupt-based input
In **polling** the CPU continuously polls the I/O in a loop, waiting for the next input. Polling is easy to program and uses software to check if data needs to be processes, however it is slow as you need to explicitly check to see if there is data that needs to be processed. Polling is also wasteful of CPU time, as the faster response time we need, the more often we need to check if that data needs to be processed. Polling also scales badly; it is difficult to build a system with many activities which can respond quickly. Note, response time depends on all other processing.
In **interrupts**, the CPU works on given tasks continuously and when an input is available it will trigger a signal to interrupt the CPU from its work to process the input data. Interrupts are fast and efficient, however can be difficult to program. Interrupts use special hardware to detect an event and run specific code held in the Interrupt Service Routine in response. Interrupts are efficient as it only runs code when necessary and fast because it is a hardware mechanism. Interrupts also scales well as the ISR response time doesn't depend on most other processing and code modules can developed independently. Interrupts provide efficient event-based processing and a response to events regardless of program state, complexity, or location. Further, it allows many multithreaded embedded systems to be responsive without an operating system.

Example : Keyboard
Polling

- CPU would continuously poll to see if a keystroke had been pressed
- If a keystroke was pressed then the information will be sent to the microcontroller which will read and process the information of the pressed key
- The CPU would then resume to continuously poll to see if a keystroke had been pressed

Interrupt

- If a key on a keyboard is pressed, then the keyboard will send an interrupt to the microcontroller
- The microcontroller will read the information of the pressed key
- The microcontroller will then return to its previous tasks

Question 2 : Describe how an interrupt service routine differs from a normal subroutine call

	Subroutine	ISR
Intended use	Separate body of code designed to perform a particular task	Software routine that is invoke in response to an interrupt
How it is invoked	Called within software (a body of code)	ISR get invoked by hardware signals from peripheral or external device

How the code is accessed	A program can access the subroutine by calling the function name of the subroutine	The code to handle interrupts are held in the Interrupt vector table
How state is preserved	The interpreter jumps back to the main program and executes the line immediately following the subroutine call Store return address in R14 Load PC with R14 to return Push all callee registers	Basic program state saved for system call The program state is reloaded and the program resumes Push R0, R1, R2, R3, R12, LR, PC, xPSR
How execution resumes where it left off after the subroutine/ISR completes	The processor is responsible for returning correctly to its activity before it was interrupted Pop all callee registers and load LR into PC	After the processor executes ISR, there is a return-from-interrupt instruction at the end and the processor resumes other code SP is back to previous value