**Lab 12 Notes: Interrupts**

**Key Terms (Cortex M based)**

* **Atomic operation:** is a process which cannot be interrupted once started.
* **Trigger**: A hardware event (the automatic transfer of software execution to a hardware event; an interrupt)
* **Process**: is the action of software as it executes
* **Thread**: The path of action a software takes as it is executed
  + **Background thread**: The execution of the interrupt service routine
  + **Foreground thread**: The execution of programs in the main() routine
  + **Multi-threaded**: foreground and background threads cooperating to perform an overall

task.

* + **Operating system:** Allows multiple foreground threads
* **EnableInterupts():** Allows interrupts to happen at this time
  + PRIMASK = 0
* **DisableInterupts():** Postpone interrupts until a later time.
  + PRIMASK = 1
* **PRIMASK:** Interrupt Priority Mask
* **BASEPRI**: Is a register for setting interrupt priority.
  + If BASEPRI is set to 3, requests level 0,1, and 2 can interrupt. Anything above 3 will be postponed. If BASEPRI is set to 0 then nothing can interrupt the current interrupt.
* **Context switch:** The steps an interrupt process routine goes through switching from **foreground** thread to a **background** thread, or when going from lower priority thread to higher priority thread.
* **Stack:** Is a specialized buffer which stores data from the top, down. As new data comes in, they push down the older data with the top being the most recent.
* **SP:** (Stack Pointer) is a register that stores the address of the last program request in a stack.
  + **MSP** (Main Stack Pointer) is the default stack pointer used by exception handlers
  + **PSP**  (Process Stack Pointer) used by user application code
* **LR:** (Link Register) holds the address to return to when a function call completes.
* **PC:** (Program Counter), also known as the Instruction Pointer(IP), is incremented after fetching an instruction. The program counter holds the memory address of the next instruction.
* **IPSR:** (Interrupt program status register) contains the exception type number of the current ISR.
* **ISR:** (Interrupt service routine), also known as an interrupt handler is a callback function who’s execution is triggered when the hardware requests an interrupt.
  + **Polled Interrupts:** One large ISR that handles all requests.
  + **Vectored Interrupts:** Many small ISRs, each with specific source of interrupts.

**Conditions for Interrupts to be executed:**

1. **Device is armed** – Allow the hardware trigger to interrupt.
2. **NVIC is enabled for the device** – Set the NVIC enable bit under 0xE000.E100 NVIC register.
3. **EnableInterupts()** – the function will set PRIMASK = 0.
4. **Has priority** – BASEPRI is set higher than the current running interrupt.
5. **Trigger** – The external hardware event that will trigger the interrupt becomes active/true

**Interrupt process routine (the context switch):**

1. Finish processing the current instruction when an interrupt trigger occurs
2. **Push registers** R0, R1, R2, R3, R12, R13 **(Stack Pointer)**, R14 **(Link Register)**, and R15 **(Program Counter)** to the **stack** (a special region in memory that stores temporary variables created by functions() --- including main() function).
3. Link register (LR) is set to **0xFFFF.FFF9** (this will return the control to the main program when the function call completes).
4. Interrupt Program Status Register (IPSR) is set to the interrupt number being processed.
5. Program Counter (PC) is loaded with the interrupt vector (address of the ISR vector).

**DO’s and DON’Ts of ISR (Interrupt Service Routine)**

* **DO** acknowledge (clear) the trigger flag
* **DO** make the ISR’s as short as possible
* **DON’T** have delay/busy wait loops in the ISR

Common Errors:

* The system will crash if the ISR doesn’t acknowledge or disarm the device requesting the interrupt.
* The ISR software should not disable

**When to use Interrupts and busy-wait**

* Busy-wait:
  + When I/O timing is predictable
  + I/O structure is simple and fixed
* Interrupts:
  + When I/O timing is variable
  + When I/O is complex
  + I/O devices exist in the system with different speeds
  + Infrequent but critical events:
    - Power failure
    - Memory faults
    - Machine error
  + Periodic interrupts:
    - Real-time clocks
    - Data Acquisition systems
    - Control systems

**Synchronizing threads**

* **Binary semaphore:** uses a shared flag system with two operations
  + **Signal:** sets the flag
  + **Wait:** check if the flag is set or not
    - **If set:** clear the flag, perform task
    - **If not set:** do nothing
  + The flag must exist as a private-global variable (using the “**static**” qualifier).
  + You must limit the access to this flag to as few modules as possible to avoid bugs.
* **The Mailbox:**