CSE 379
Lab #5
Spring 2022

Objective
In this lab, you will learn how to service interrupts on the ARM processor.

Description
Write interrupt initialization and handler routines to process two interrupts. The first interrupts the processor when the user hits a key on the keyboard. The initialization routine for this interrupt should be called *uart_interrupt_init*, and it should generate an interrupt when a keystroke is generated by the user in *PuTTy* (UART0). The interrupt handler associated with this interrupt is *UART0_Handler*. The second interrupts the processor when the button labeled SW1 on the Tiva board is pressed. The initialization routine for this interrupt should be called *gpio_interrupt_init*, and it should generate an interrupt when a SW1 on the Tiva board is pressed. The interrupt handler associated with this interrupt is *Switch_Handler*. In both handlers the interrupt should be cleared at the beginning of the routine, and control transferred back to your program after the interrupt is processed. Your main program (*lab5* routine) should display the number of times SW1 was pressed and the number of times a key on the keyboard was pressed. This information should be continuously updated until the user hits *q* (lowercase) on the keyboard. Polling may NOT be used to obtain data from the keyboard or the switches. The data must be obtained through the use of interrupts. When your lab is graded, a breakpoint will be set in each handler to verify that interrupts are working properly and that the handler is processing the button press and keypress.

Startup Code
Use the startup code on the *Labs* page of the course website, NOT the startup assembly file that you've been including in your project when you create a project in past labs.

PreLab Flowchart
A flowchart of your main program and for each handler before you will be allowed to work in Bonner 114.

Skeleton Code
The following skeleton code shown below can be used for *lab_5.s*.

```
        .data

        .global prompt
        .global mydata

prompt:     .string "Your prompt with instructions is place here", 0
mydata:     .byte 0x20  ; This is where you can store data.
                        ; The .byte assembler directive stores a byte
                        ; (initialized to 0x20) at the label mydata.
                        ; Halfwords & Words can be stored using the
                        ; directives .half & .word

        .text

        .global uart_interrupt_init
        .global gpio_interrupt_init
        .global UART0_Handler
```

```
      .global Switch_Handler
      .global Timer_Handler         ; This is needed for Lab #6
      .global simple_read_character
      .global output_character      ; This is from your Lab #4 Library
      .global read_string           ; This is from your Lab #4 Library
      .global output_string         ; This is from your Lab #4 Library
      .global uart_init       ; This is from your Lab #4 Library
      .global lab5

ptr_to_prompt:          .word prompt
ptr_to_mydata:          .word mydata

lab5: ; This is your main routine which is called from your C wrapper
      PUSH {lr}               ; Store lr to stack
      ldr r4, ptr_to_prompt
      ldr r5, ptr_to_mydata

      bl uart_init
      bl uart_interrupt_init
      bl uart_interrupt_init

      ; This is where you should implement a loop, waiting for the user to
      ; enter a q, indicating they want to end the program.

      POP {lr}          ; Restore lr from the stack
      MOV pc, lr



uart_interrupt_init:

      ; Your code to initialize the UART0 interrupt goes here

      MOV pc, lr


gpio_interrupt_init:

      ; Your code to initialize the SW1 interrupt goes here
      ; Don't forget to follow the procedure you followed in Lab #4
      ; to initialize SW1.

      MOV pc, lr


UART0_Handler:

      ; Your code for your UART handler goes here.
      ; Remember to preserver registers r4-r11 by pushing then popping
      ; them to & from the stack at the beginning & end of the handler

      BX lr             ; Return


Switch_Handler:

      ; Your code for your UART handler goes here.
```

```
      ; Remember to preserver registers r4-r11 by pushing then popping
      ; them to & from the stack at the beginning & end of the handler

      BX lr             ; Return


Timer_Handler:

      ; Your code for your Timer handler goes here.  It is not needed
      ; for Lab #5, but will be used in Lab #6.  It is referenced here
      ; because the interrupt enabled startup code has declared
Timer_Handler.
      ; This will allow you to not have to redownload startup code for
      ; Lab #6.  Instead, you can use the same startup code as for Lab #5.
      ; Remember to preserver registers r4-r11 by pushing then popping
      ; them to & from the stack at the beginning & end of the handler.

      BX lr             ; Return


simple_read_character:

      MOV PC,LR         ; Return

      .end
```


Partners

You will work with a partner in this lab. You will be instructed in your regularly scheduled lab session or via e-mail if your partner will be the same as the one you worked with on Labs #3 and #4.