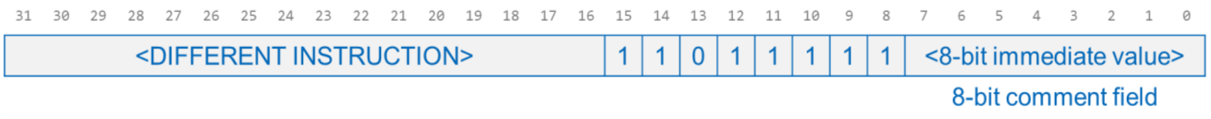
Lab #05

Toggling Lights using SuperVisor Calls

Assignment Type:	Individual
Assignment Duration:	1 week
Marks (of course total):	3%
Submission Method:	https://tcd.blackboard.com
Submission Deadline:	23:59 on Friday the 25th of February 2022

Introduction

This lab will introduce you to the concept of setting up and using SuperVisor Call (SVC) exceptions. The code can be run on the Wokwi online simulator for the Pi Pico or on the Raspberry Pi Pico hardware platform however it is strongly advised that you use the actual hardware platform with the low-level debugger to make development easier.



```
#include "hardware/regs/addressmap.h"
#include "hardware/regs/m0plus.h"

.syntax unified                 @ Specify unified assembly syntax
.cpu    cortex-m0plus           @ Specify CPU type is Cortex M0+
.thumb                          @ Specify thumb assembly for RP2040
.global main_asm                @ Provide program starting address to the linker
.align 4                        @ Specify code alignment

.equ    SLEEP_TIME, 500         @ Specify the sleep time (in ms)
.equ    LED_GPIO_PIN, 25        @ Specify the pin that the LED is connected to
.equ    LED_GPIO_OUT, 1         @ Specify the direction of the GPIO pin
.equ    LED_VALUE_ON, 1         @ Specify the value that turns the LED "on"
.equ    LED_VALUE_OFF, 0        @ Specify the value that turns the LED "off"
.equ    SVC_ISR_OFFSET, 0x2C    @ The SVC is entry 11 in the vector table
.equ    SVC_MAX_INSTRS, 0x01    @ Maximum allowed SVC subroutines

@ Entry point to the ASM portion of the program
main_asm:
    bl      init_gpio_led       @ Initialise the GPIO LED pin
    bl      install_svc_isr     @ Install the SVC interrupt service routine
loop:
    svc     #0                  @ Call the SVC ISR with value 0 (turns on LED)
    nop                         @ Add a no-op instruction for alignment after SVC
    bl      do_sleep            @ Short pause before proceeding
    svc     #1                  @ Call the SVC ISR with value 1 (turns off LED)
    nop                         @ Add a no-op instruction for alignment after SVC
    bl      do_sleep            @ Add a short pause before proceeding
    b       loop                @ Always jump back to the start of the loop

@ Subroutine used to introduce a short delay in the application
```

```
do_sleep:
    <TODO – add assembly code to implement the sleep delay using sleep_ms>

@ Subroutine used to initialise the PI Pico built-in LED
init_gpio_led:
    <TODO – add assembly code to initialise the LED GPIO pin>

@ Subroutine used to install the SVC interrupt service handler
install_svc_isr:
    ldr     r2, =(PPB_BASE + M0PLUS_VTOR_OFFSET)    @ <TODO – add comment>
    ldr     r1, [r2]                                @ <TODO – add comment>
    movs    r2, #SVC_ISR_OFFSET                     @ <TODO – add comment>
    add     r2, r1                                  @ <TODO – add comment>
    ldr     r0, =svc_isr                            @ <TODO – add comment>
    str     r0, [r2]                                @ <TODO – add comment>
    bx      lr                                      @ <TODO – add comment>

@ SVC interrupt service handler routine
.thumb_func                     @ Required for all interrupt service routines
svc_isr:
    push    {lr}                @ <TODO – add comment>
    ldr     r0, [sp, #0x1C]     @ <TODO – add comment>
    subs    r0, #0x2            @ <TODO – add comment>
    ldr     r0, [r0]            @ <TODO – add comment>
    ldr     r1, =#0xFF          @ <TODO – add comment>
    ands    r0, r1              @ <TODO – add comment>
    cmp     r0, #SVC_MAX_INSTRS @ <TODO – add comment>
    bgt     svc_done            @ <TODO – add comment>
    adr     r1, svc_jmptbl      @ <TODO – add comment>
    lsls    r0, #2              @ <TODO – add comment>
    ldr     r1, [r1, r0]        @ <TODO – add comment>
    mov     pc, r1              @ <TODO – add comment>
svc_done:
    pop     {pc}                @ <TODO – add comment>

@ First function of SVC subroutine - turn on the LED
svc_num0:
    <TODO – add assembly code to turn on the LED>
    b       svc_done            @ Branch back to the main ISR when done

@ Second function of SVC subroutine - turn off the LED
svc_num1:
    <TODO – add assembly code to turn off the LED>
    b       svc_done            @ Branch back to the main ISR when done

@ SVC function entry jump table.
.align 2
svc_jmptbl:
    .word svc_num0              @ Entry zero goes to SVC function #0.
    .word svc_num1              @ Entry one goes to SVC function #1.
    .word 0                     @ Null termination of the jump table.

@ Set data alignment
.data
    .align 4
```

After completing this lab, you should have successfully created and compiled assembly code to setup an exception handler for SVC calls and implemented two SVC call handlers, the first to turn on the Raspberry Pi Pico built-in LED and the second to turn it off. Your code should alternately flash the LED on-and-off after a short delay.

Instructions

1. Make sure that your "pico-apps" repository is up to date before starting the lab.
2. Copy the code from "examples/blink_asm/blink_asm.c" into "labs/lab05/lab05.c" in the *"labs/lab05"* template folder.
3. Copy the skeleton ARM ASM code from this exercise into "labs/lab05/lab05.S" in the "labs/lab05" template folder.
4. Complete the missing code marked as <TODO> for the following subroutines:
   - **do_sleep**, **init_gpio_led**, **svc_num0**, **svc_num1**
5. Complete the missing comments marked as <TODO> for the following subroutines:
   - **install_svc_isr**, **svc_isr**
6. Make sure that your code is functional (the built-in LED on the Raspberry Pi Pico should flash on-and-off with a 500ms delay between each toggle).
7. Commit the changes you have made and push them back to your "*pico-apps*" main repository

Grading Scheme

Complete all steps in the instructions section and then upload the completed "**lab05.elf**", "**lab05.uf2**", "**lab05.c**" and the "**lab05.S**" files to the lab05 assignment in Blackboard before the specified deadline to complete the lab. This lab is worth a total of 3% of your year-end results for the CSU232021 module and the marking scheme is as follows.

	INCOMPLETE [0%]	COMPETENT [50%]	PROFICIENT [100%]
SUBMISSION [10%]	Student has failed to submit the lab exercise.	Student has submitted the lab exercise but some of the required deliverables are missing.	Student has submitted the lab exercise and has included all the required deliverables.
IMPLEMENTATION [65%]	Student has not made a sufficient attempt at completing the lab exercise.	Student has completed the lab exercise however the implementation does not work as expected.	Student has completed the lab exercise and the implementation works as expected.
QUALITY [25%]	The code is badly structured or has not been well commented. Some or all of the specified tasks were not completed.	The code is adequately structured and there are some comments present. Some of the specified tasks were not completed.	The code is well structured with good and clear commenting throughout. All of the specified tasks were completed.