Lab 2: Answer Sheet

Student Names: Kyler Fillerup, Casey Nordgran
Student uIDs: u0862975, u0887369
Student Class Section: 4
Partner Group #: 32

In-Lab Assignment

Question 1. What does the "My First Assembly File" application do?

It enters the main loop, with the only command being to branch back to the main label. In other words, it is an infinite loop.

Question 2. Can you tell in the debugger if you are still counting, or already in the busy loop? How? Write your new main code in the answer sheet.

Yes, you can step through the program and check the register (info registers) to check the value of the register to see if it is less than 100. If the register value is at 100 ,the program execution is then in the busy loop. You can also step through each instruction and see if you are stuck in the loop.

```
.equ  STACK_TOP, 0x20000800

        .section .int_vector,"a",%progbits @ First linker code section
        .global _start                       @ Linker entry point
_start:
        .word STACK_TOP                    @ Stack pointer
        .word main                         @ program counter start address
                                           @ End of int_vector section

        .text
        .syntax unified
        .type main, %function
main:
            mov r0, #0
loop:
            add r0, r0, #1
            cmp r0, #100
            bne loop
busy:
        b busy
.end
```

Question 3. Explain the difference between the GDB commands step and stepi. Use the GDB help command to find the difference.

The step command steps through the program until it reaches a different source line that has debugging information. Therefore when using step, if your in a function that doesn't have debugging info, then it will proceed until a line is reached with debug info. The stepi command steps through exactly one instruction then returns to debugger, regardless of whether there is debug info or not.

Question 4. Disassemble the main.o file. Do you see the function call into your gpio.o file? Explain.
Yes, under the 'main' section the call to initGPIO & setGPIO appears on lines 20 & 28 as follows.
20: f7ff fffe bl 0 <initGPIO>
28: f7ff fffe bl 0 <setGPIO>

Question 5. In the left window pane, Project Explorer, open Lab2 ! Debug ! Lab2.lst file. Does it look familiar? Explain the contents of the file.
Yes, it appears to be very similar to the terminal printout in the command window when 'objdump' is invoked. The Lab2.lst appears to be a little more compact with info, and shows other debug info above. Other than that is is very much like a disassembled object file.

Post-Lab Assignment
Question 6. Write the initGPIO and setGPIO assembly functions in gpio.s. Use main.s as an example for modifying gpio.s. Comments were provided in the function stubs suggesting content. Reread section 5 before writing code.

Added to main.s

```
        mov      r0, #24
        bl       initGPIO   @ Call initGPIO in gpio.s to initialize GPIO 24

        mov      r0, #0
        bl       setGPIO    @ Call setGPIO in gpio.s to write 0 to GPIO output register

        mvn     r0, #1
        ror     r0, #1      @ set r0 to all 1's except for pin 31
        add     r4, r0, #0  @ r4 is set to all 1's except pin 23
        ror     r4, #8      @ r4 is used as reference to known when
                            @ to reset the zero in r0 back to pin 31
loop:
        bl      setGPIO     @ enter setGPIO
        ror     r0, #1      @ rotate the all bits right one
        cmp     r0, r4
        bne     loop
        ror     r0, #23     @ if r0 has low bit on pin 23, rst to pin 31
        b       loop
.end
```

Added to gpio.s

```
initGPIO:
        @ Load GPIO_OUT_BASE address
            movw   r1, #:lower16:GPIO_OUT_BASE
            movt   r1, #:upper16:GPIO_OUT_BASE
        @ Calculate the GPIOx register address
            lsl    r0, 2
            add    r1, r0

        @ Write 1 to config register to set GPIO as output
            mov    r2, #0x1
            str    r2, [r1, #0]

            .rept 7      @ repeat block increments address by 4,
            add    r1, #4       @ and configs next GPIO to output
            str    r2, [r1, #0]
            .endr
            bx     lr     @ Return
```

```
@ Set the value of all 32 GPIO output bits based on the input bits
@ Inputs: 32bit value is provided in r0
@ Output:
        .global setGPIO
        .type   setGPIO, %function
setGPIO:
        @ Load GPIO_OUT register address
            movw   r1, #:lower16:GPIO_OUT
            movt   r1, #:upper16:GPIO_OUT
        @ Write 32bit value to GPIO output register
            str    r0, [r1, #0]

            mov    r2, #1000
delayloop:                    @  nested loops to add delay
            mov    r3, #1000
delayloop2:
            subs   r3, #1
            bne    delayloop2
            subs   r2, #1
            bne    delayloop
            bx     lr  @return
```

Question 7. Is LED 1 really permanently turned on? Verify using an oscilloscope to see if LED 1 is really turned on continuously. What is the frequency of the signal that you see? How does this related to the CPU clock frequency?

Connecting the oscilloscope ground terminal to the board ground, and then probing just between the LED and the resistor in series after it, the scope showed the LED to be a signal frequency of turning on and off at about 2.29MHz. As the LED status is not the only instruction in the loop, this frequency is a portion of the circuits clock frequency, where the other clock cycles are coming from the other instructions in the loop.

Question 8. First, initalize all 8 LEDs (GPIO 24-31). Have only one LED on at a time but round-robin through all the LEDs. Add enough delay between each LED transition such that it is visible to the Human eye. Demo it.

8 LED Round-robin demonstration works great. Checked off by TA