CS250 Lab 05 (Take-Home)

1 RGB Color Blender

Whenever you write high voltage to a GPIO pin, the pin will carry a constant voltage of 3.3V. The hardware alone cannot be used to slightly increase/decrease the brightness of an LED, for example. For this task, you will make use of a pulse-width modulation (PWM) library on the Pi to change the pulse sent to an LED.

To use this library, simply include softPwm.h in your C code. Study the file pwm_api.c included in your in-lab download to view the API for the PWM library. You will need the functions described there to complete this task.

Your task is to implement an RGB color blender. You will wire five switches to your breadboard (one for each of red, green, blue, increase brightness, and decrease brightness) as well as your multicolor LED. The pins on the multicolor LED (pictured right) from left to right are: **blue, green, ground, red**.

When you hold down the brightness button and then press a color button, that color in the LED should be increased by 5 brightness on a scale of 0-100. Similarly, if the decrease brightness button is held and a color button is pressed, the LED should decrease by 5 brightness.

Once a color reaches 100 brightness, your program should not continue increasing the brightness of that color. Doing so could damage your LED. Also, to protect your hardware components, make sure to unplug your breadboard's power before doing any rewiring.

To see what your solution should look like, you can watch a demo here:
`https://goo.gl/FoWVQj`

Write your code in rgb.c and compile it by typing the following to link the proper libraries:

```
gcc -std=c99 -o rgb rgb.c -lwiringPi -lpthread
```

2 ARM Assembly

In this section, you will begin writing ARM assembly programs. Open the file template.s provided to you to view a template that you can use for all of your programs. The program only executes an empty command and then exits.

2.1 Using GDB

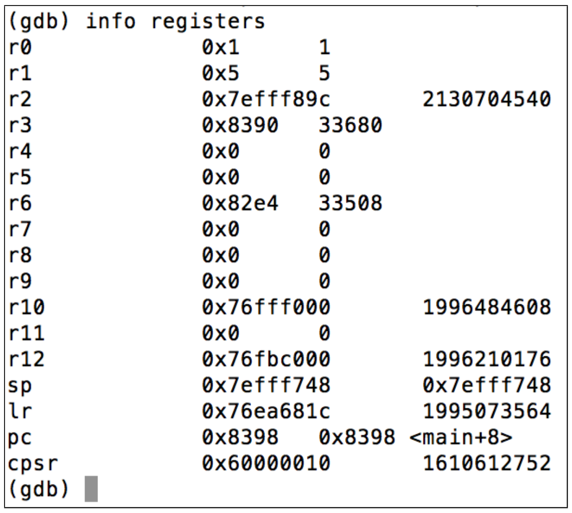
We will begin by using the GNU Debugger (GDB) that you should be familiar with from CS240. To use GDB on a program, you need to invoke gcc with an extra flag, then run GDB:

```
gcc -gstabs -o gdb_example gdb_example.s
gdb gdb_example
```

Inside GDB, run these instructions:

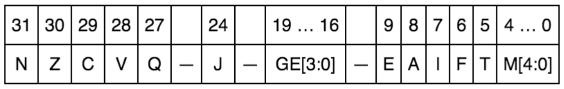
```
break main
run
```

These set a break point at the beginning of main and start the execution of your program inside GDB. Now type "next" and "info registers" and you should see this:



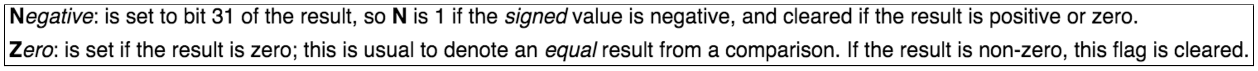
```
(gdb) info registers
r0             0x1      1
r1             0x5      5
r2             0x7efff89c       2130704540
r3             0x8390   33680
r4             0x0      0
r5             0x0      0
r6             0x82e4   33508
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x76fff000       1996484608
r11            0x0      0
r12            0x76fbc000       1996210176
sp             0x7efff748       0x7efff748
lr             0x76ea681c       1995073564
pc             0x8398   0x8398 <main+8>
cpsr           0x60000010       1610612752
(gdb)
```

Notice how the first line of the program stored 5 in register r1. In this example, we will focus on the cpsr register. To print the value of cpsr, type "print /t $cpsr", where /t means print as a binary integer. If you print the value of cpsr now, you will notice that it only prints 31 bits instead of 32.



31	30	29	28	27		24		19 … 16		9	8	7	6	5	4 … 0
N	Z	C	V	Q	—	J	—	GE[3:0]	—	E	A	I	F	T	M[4:0]

In fact, according to the diagram of the 32 bits of cpsr above, the value of "N" is missing. It turns out that this is not present because we have not evaluated any comparisons yet (e.g., is the value in register r1 less than 4?). Here are the descriptions of the left-most two flags, which will be important in this section:



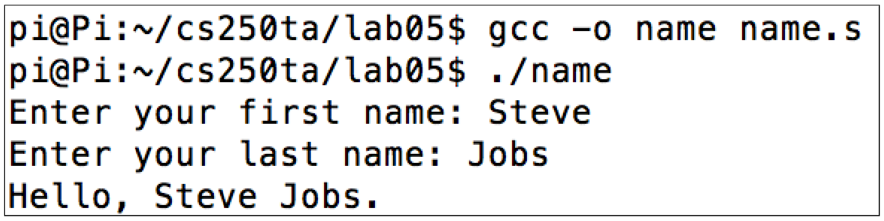
Negative: is set to bit 31 of the result, so **N** is 1 if the *signed* value is negative, and cleared if the result is positive or zero.
Zero: is set if the result is zero; this is usual to denote an *equal* result from a comparison. If the result is non-zero, this flag is cleared.

Continue stepping through the program in GDB, and after each call to "cmp" (which in this case repeatedly compares register r4 to the number 4) print cpsr. This should give you a good idea how comparisons work in ARM.

Task: Your graded task for this section is to take a screenshot where you have printed the value of cpsr at any point after the first comparison in this program. Identify on the screenshot where the N and Z bits are and why the N and Z bits have the value that they do.

2.2 name.s

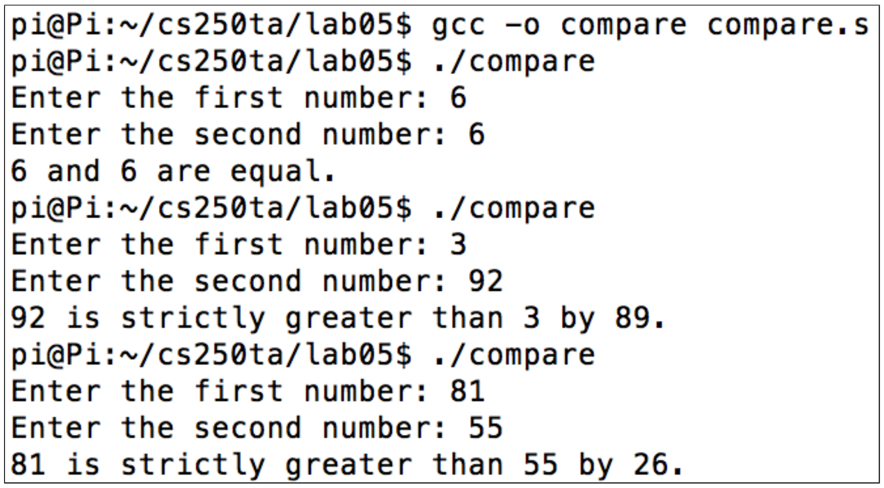
Task: Create a program hello.s that reads the user's first name from stdin, then reads their last name from stdin, then prints "Hello, firstName lastName.". If you find that your output will not work with the testall program (read the grading section below for info on the testall), add the commands "mov r0 $0" and "bl fflush" right at the end of your program to flush your output to stdout.



```
pi@Pi:~/cs250ta/lab05$ gcc -o name name.s
pi@Pi:~/cs250ta/lab05$ ./name
Enter your first name: Steve
Enter your last name: Jobs
Hello, Steve Jobs.
```

2.3 compare.s

Task: Create a program compare.s that reads in a number x from stdin, then reads in a second number y from stdin. If the numbers are equal, print "x and y are equal."; otherwise, print "$max\{x, y\}$ is strictly greater than $min\{x, y\}$ by $|x - y|$.".



```
pi@Pi:~/cs250ta/lab05$ gcc -o compare compare.s
pi@Pi:~/cs250ta/lab05$ ./compare
Enter the first number: 6
Enter the second number: 6
6 and 6 are equal.
pi@Pi:~/cs250ta/lab05$ ./compare
Enter the first number: 3
Enter the second number: 92
92 is strictly greater than 3 by 89.
pi@Pi:~/cs250ta/lab05$ ./compare
Enter the first number: 81
Enter the second number: 55
81 is strictly greater than 55 by 26.
```

3 Grading

You have been provided with a testall file (type "./testall" in the terminal to check your grade on name.s and compare.s) to check the validity of your two ARM programs. rgb.c will be graded in person at the beginning of next week's lab since there is a breadboard component. Your GDB screenshot will also be collected during lab next week. Check the grading rubric for the points breakdown.