CSCI 2500 — Computer Organization
Lab 04 (document version 1.1)

- This lab is due by the end of your lab session on Wednesday, September 26, 2018.
- This lab is to be completed **individually**. Do not share your code with anyone else.
- You **must** show your code and your solutions to a TA or mentor to receive credit for each checkpoint.
- Labs are available by 6:00PM on Mondays before your lab sessions. Plan to start each lab early and ask questions during office hours, in the discussion forum on Submitty, and during your lab session.

1. **Checkpoint 1:** Start by installing the latest version of SPIM, which is a MIPS simulator. The URL is: `https://sourceforge.net/projects/spimsimulator/files/`.

   After installing SPIM, download the `helloworld.s` code from within Course Materials in Submitty, then run the code in SPIM. Within the `spim` interface, enter the '`?`' character to get help in loading a program, running it, stepping through it, etc. Of particular interest are the `run`, `step`, and `print_all_regs` commands.

   Also, in general, refer to Appendix A of our textbook and use this URL as a reference for how to write MIPS programs for this environment:
   `http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm`.

   And refer to the "MIPS Reference Data Card" from your textbook, also available within Course Materials in Submitty.

   Modify the given `helloworld.s` code by changing the "Hello world" string to instead be "MIPS is awesome!" (and add a newline to the end of the output).

   Take the time to understand how this example program works. Be sure you understand how to set up the system call (i.e., `syscall`). In particular, what happens if you load register `$v0` with `1` instead of `4`? In other words, describe what happens when you change the first `main` line to:

   ```
   main:  li $v0, 1       # syscall 4 (print_str)
   ```

   What happens when you change the above value to `2`? To `3`? And so on?

   Finally, also use this checkpoint to get familiar with the SPIM interface.

2. **Checkpoint 2:** For the second checkpoint, you will translate the following block of high-level code into MIPS:

```
{
  int x = 0;

  if ( x < 5 )
  {
    x += 3;
  }
}
```

To accomplish this, associate variable `x` with register `$t0`. In other words, register `$t0` will hold the value of `x`. After you implement this logic in MIPS, use MIPS code to display result `x` using the following output format:

```
x ($t0) equals <value-of-x>
```

As an example, the above code example should output the following:

```
x ($t0) equals 3
```

You will need to make use of the `print_string` and `print_int` system calls, as well as the Set Less Than (`slt`) and Branch On Equal (`beq`) instructions.

Test your MIPS code by changing the initial value of `$t0` to `2`, `4`, `5`, `6`, `100`, etc.

3. **Checkpoint 3:** For the third checkpoint, we will revisit the GCD algorithm, but this time implement it using MIPS. More specifically, in MIPS, implement the GCD algorithm that uses subtraction instead of the "mod" operator.

To complete this checkpoint, use registers `$t0` and `$t1` as inputs to your GCD code. You can hard-code these two values using Load Immediate (`li`) instructions. Make sure you use positive values less than `32768`.

Finally, display the resulting GCD of the two given values. Use the format shown below:

```
GCD( <first-value>, <second-value> ) is <GCD-result>
```

As an example, if values `$t0` and `$t1` are `45` and `54`, respectively, the output is:

```
GCD( 45, 54 ) is 9
```

Try it again with inputs `54` and `45`; in other words, make sure it works if you swap the two inputs!