Homework #2 **Due: 9/16**

In this assignment, you will implement a MIPS program that does rational arithmetic. In the computer, rationals are represented by two 32-bit `int`s, numerator and denominator in that order, stored in successive memory locations when in memory and in successive registers when being used for computation. By convention, the numerator carries the sign, so the denominator is unsigned.

You will build a set of functions that perform this arithmetic. By convention, all of them leave results in `($v0, $v1)`, representing the rational number `$v0 / $v1`.

You will find a tarball containing a starter file `rational.s` on the course web page. This is the only file you need to modify to complete this assignment. Do not change any other files in the tarball.

`rational.s` contains a `rat_print()` function you can use as an example and to print out your test results. Do not delete or alter this function except as directed below: It will be called by the grader's tests.

Be sure to follow the register usage conventions. The file `mips_register_conventions.pdf`, downloadable from the Unit 2 hyperlink on the Schedule (and handed out in class), may be helpful with this.

It is suggested that you build the program in the order indicated to make testing and debugging convenient. Be sure to do adequate testing!

Do not worry about detecting overflow (although the last problem will reduce the chance of this).

Put your test program (`main:`) in `main.s`. Remember that the grader will use a different `main.s` for their tests and will only refer to yours if those tests fail in order to assign partial credit.

You only need to submit hard copy of your `rational.s` listing, since that's the only one you should modify.

Remember that the sign should be carried in the numerator.

1. [25 points] Add a function `rat_mul()` that takes one rational number `($a0, $a1)` and multiplies it by `($a2, $a3)` using the rules of rational arithmetic.

2. [10 points] Add a function `rat_div()` that takes one rational number `($a0, $a1)` and divides it by `($a2, $a3)` using the rules of rational arithmetic. `rat_div()` should do its work by calling `rat_mul()` after making a small alteration to its arguments.

   Don't worry about divide-by-zero.

3. [25 points] Add a function `rat_add()` that takes one rational number `($a0, $a1)` and adds it to `($a2, $a3)` using the rules of rational arithmetic.

4. [10 points] Add a function `rat_sub()` that takes one rational number `($a0, $a1)` and subtracts `($a2, $a3)` from it using the rules of rational arithmetic. `rat_sub()` should do its work by calling `rat_add()` after making a small alteration to its arguments.

5. [30 points] Rational numbers should always be simplified. For instance, "4/8" should always be simplified to "1/2". This doesn't just look good, but reduces the chance for overflow. To do this, divide the numerator and denominator of the rational by their greatest common denominator (GCD). Remember that dividing a number by its GCD with another number always (by definition) leaves a zero remainder.

   The tarball contains a file `gcd.s` that provides a function `gcd()` that will compute the GCD of arguments `$a0` and `$a1` and return the result in `$v0`.

   Write a `rat_simplify()` function to simplify its input (in `$a0` and `$a1` using `gcd()` and make all of the functions in problems 1-4 call it to simplify their results just before they return. Also modify `rat_print()` to simplify its input before printing it.