CSCI-21 Programming Assignment #2. Due 2/25/19

For these programming exercises, you may use only these instructions or macros:

```
add addi addiu addu and andi div divu li lui lw mfhi mflo
  mult multu nor or ori sll sra srl sub subu sw xor xori
```

Except for exercise 4, in the Settings menu of QtSpim set Bare Machine OFF, Allow Pseudo Instructions ON, Load Exception Handler OFF, Delayed Branches OFF, Delayed Loads ON, Mapped IO OFF, Mapped IO OFF. For exercise 4 **ONLY**, set Bare Machine ON and Allow Pseudo Instructions OFF.

Run the programs by setting the value of the PC to 0x00400000 and then single stepping (pushing F10) or by multiple stepping (push F11 and enter a number of steps), observing the results in the SPIM register display window.

Exercise 1:

Write a program that adds up the following integers and leaves the answer in register $t0. Use `li` instructions or macros to set (different) registers to the values before adding:

```
 456
-229
 325
-552
```

In the notes to load immediate each instruction, indicate whether the loads are actually instructions or if they are macros. Also indicate how you could tell what determined which was which?

Exercise 2 - Shifting and Adding:

Initialize register $t0 (will hold the sum) to zero. Then add 4096_{10} to $t0 sixteen times. You don't know how to loop yet, so do this by making 16 copies of the same instruction. The hexadecimal value of 4096_{10} is 0x1000.

Next, initialize register $t1 to 4096_{10}. Shift $t1 left by the correct number of positions so that registers $t0 and $t1 contain the same bit pattern.

Finally, initialize register $t2 to 4096_{10}. Add $t2 to **itself** the correct number of times so that it contains the same bit pattern as the other two registers.

Exercise 3 - Signed and Unsigned Adding:

Initialize register $t1 to 0x7000. Shift the bit pattern so that $t1 contains 0x70000000. Now use the addu instruction to add $t1 to itself. Is the result correct?

Then repeat the procedure using register $t2 and the add instruction. What happens?

Write a short description (2 or 3 sentences is enough) of the differences in a text file called P2Ex3.txt and submit it with your other files.

Exercise 4 – Expression Evaluation (for this one disallow macros!):

Write a program that determines the value of this expression: (x*y)/z

Use x = 1600000_{10} (=0x00186A00), y = 80000_{10} (=0x00013880), and z = 400000_{10} (=0x00061A80). Initialize three registers to these values. Since the immediate operand of the ori instruction is only 16 bits wide, use shift instructions to move bits into the correct locations of the registers. Do not use any macros for this program. This problem is somewhat tricky, because the expression (x*y) will have a result that is bigger than 32 bits long. Therefore, choose the order of multiply and divide operations so that the significant bits always remain in the lo result register. **You will have to work this out algebraically first.**

Exercise 5:

Evaluate the polynomial:

$2x^3 - 3x^2 + 5x + 12$

Use symbolic addresses for variables `x` and `answer` (holds the result). Assume that the value in `x` is small enough so that all results fit into 32 bits. Since load delays are turned on in SPIM be careful what instructions are placed in the load delay slot.

Verify that the program works by using several initial values for x. Use x = 0 and x = 1 to start since this will make debugging easy. Then try some other values, such as x = 10 and x = -1.

Write the program following the hardware rule that two or more instructions must follow a `mflo` instruction before another `mult` instruction (the delay slot). Try to put useful instructions in the delay slots that follow the mflo.

Exercise 6:

Evaluate the expression:

18xy + 12x - 6y + 12

Use symbolic addresses for x, y, and the answer. Assume that the values are small enough so that all results fit into 32 bits. Since load delays are turned on in SPIM be careful what instructions are placed in the delay slots.

Verify that the program works by using several initial values for x and y (for now, just modify and re-run the program). Use x = 0, y = 1 and x = 1, y = 0 to start since this will make debugging easy. Then try some other values.