CSCI-21 Assignment #3, due 10/10/22

These exercises call for using branch macros. Some branch macros do not work correctly in QtSpim. If so, turn off branch delays. If they work for you, turn on branch delays. Write these as three programs.

Exercise 1 – a counting loop
Write a program that computes the sum of the integers from 1 to 100 by addition:

1 + 2 + 3 + 4 + 5 + ... + 98 + 99 + 100

Do this by using the `blt` macro (branch if less than) described in appendix D. Before the loop, initialize register `$t0` to zero to contain the sum, initialize another register to one (to be the counter), and another register to 101 to mark the end of the sequence. Inside the loop, add the counter to the sum, increment the counter and `blt` back to the top of the loop if the counter is less than 101.

Single-step the program until you get a good idea of how it is working. Then let the program run. To stop the program, use `syscall` with the code 10 in `$v0`. See page 41 in the text for a table of the syscall codes.

Exercise 2 – more counting
Extend your program from exercise 1 to compute three sums:

1 + 2 + 3 + 4 + ... + 99 + 100 # all values 1…100

1 + 3 + 5 + 7 + ... + 97 + 99 # all odd values 1…100

2 + 4 + 6 + 8 + ... + 98 + 100 # all even values 1…100

Use register `$t0` for the sum of evens, register `$t1` for the sum of odds, and register `$t2` for the sum of all the numbers.

Do this with only one counting loop. The loop body will contain the logic to add the counter to the proper sums.

Do not use division to check odd or even, it is not necessary. A bitwise instruction is enough.

Again, stop the program with a `syscall` with 10 in `$v0`.

Exercise 3 – Fibonacci Series
Write a program that computes terms of the Fibonacci series, defined as:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ... (This is sometimes started with a 0^{th} term: 0, 1, 1, 2, etc.)

The first two terms in the series are fixed, and each subsequent term in the series is the sum of the preceding two terms. So, for example, the term 13 is the sum of the terms 5 and 8.

Write the program as a counting loop that terminates when the 100th term of the series has been computed. Use a register for the current term and a register for the previous term. Each execution of the loop computes a new current term and then copies the old current term to the previous term register. Count the number of passes to stop after the 100th term. Do not worry about overflow: this will FAR exceed the capacity of a register to hold the value.

Notice how few machine language instruction this program takes. It would be hard for a compiler to produce such compact code from a program in a high level language. (Of course, your program is not doing any IO.) Again, stop the program with a `syscall` with 10 in `$v0`.

None of the programs will do any I/O, they are just looping exercises.