You’ll need to use a text editor — create “lab1.s” and enter the following code. Once you’ve got it typed in, start the qtspim simulator, and load the code in. Single step through the code, and make sure you understand what’s happening in each of the registers, and how it’s all working...

```assembly
# lab1.s
.text
main: li $s0, 0 # Sum
    li $s1, 0 # Index
    li $s2, 10 # Limit of 10

loop: beq $s1, $s2, done
    add $s0, $s0, $s1
    addi $s1, $s1, 1
    j loop

done: jr $ra
```

Once you’ve got that working...
1) Create lab2.s, a modification where you calculate the sum of the numbers 0 to 20 inclusive. Yes, it should be easy to do...

2) Create lab3.s, where you calculate the sum of the EVEN numbers from 0 to 20 inclusive. This should be easy too…. Rather than incrementing by 1, you could increment by 2.

3) And one final slightly more tricky thing… Create lab4.s, where you calculate the sum of the squares of i, from 0 to 20 inclusive. In other words, rather than

   ```c
   sum = sum + i;
   ```

you’ll have

   ```c
   sum = sum + i * i;
   ```

In MIPS, if you multiply two 32-bit registers, you’ll get a 64-bit result, saved in special registers called LO and HI (low being the low 32 bits). mflo moves the value in the low register to a register that you can use for ordinary math, while mfhi does the same for the high register. You might want something like this in your code:

```assembly
mult $s1, $s1
mflo $t0  # The low 32 bits of s1*s1 is now in temp register $t0
```

Once you’ve found the sum of the squares, from 0 to 20 inclusive, show your TA!
Part 2... Let’s take a quick look at some x86 code! Enter the C program into sum.c. You can run your program with

```bash
% gcc sum.c -o sum
% ./sum
```

Then, run the C compiler, and have it produce assembly language output.

```bash
% gcc sum.c -S
```

The C code. Type this in. Don’t try to copy and paste, because it won’t quite work right. Yes, we’ll know if you tried to copy and paste. Get used to typing on the keyboard, because you’re going to do a lot of it.

```c
#include <stdio.h>

int main()
{
    int sum, i;
    i = 0;
    for (i = 0; i < 10; i = i + 1)
        sum = sum + i;
    printf(“The sum is %d\n”, sum);
}
```

Assembly code in “sum.s” after running gcc sum.c -S

You should be able to spot the instructions that do math, and the branch and jump instructions....

```assembly
.globl main
.type main, @function
main:
.LFB0:
    .cfi_startproc
    pushq  %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq%rsp, %rbp
    .cfi_def_cfa_register 6
    movl$0, -8(%rbp)
    addl$1, -8(%rbp)
    jmp  .L2
.L3:
    movl-8(%rbp), %eax
    addl%eax, -4(%rbp)
    addl$1, -8(%rbp)
.L2:
    cmpl$9, -8(%rbp)
    jle  .L3
    movl$0, %eax
    popq%rbp
    .cfi_def_cfa_register 6
    movl$0, -8(%rbp)
    addl$1, -8(%rbp)
.L3:
    movl-8(%rbp), %eax
    addl%eax, -4(%rbp)
    addl$1, -8(%rbp)
    cmpl$9, -8(%rbp)
    jle  .L3
    movl$0, %eax
    popq%rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq%rsp, %rbp
    .cfi_def_cfa_register 6
    movl$0, -8(%rbp)
    movl$0, -8(%rbp)
    jmp  .L2
.L2:
    cmpl$9, -8(%rbp)
    jle  .L3
    movl$0, %eax
    popq%rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size main, -.main
.ident "GCC: (Debian 8.3.0-6) 8.3.0"
.section .note.GNU-stack,"",@progbits