11.) Convert the following C-like code into MIPS assembly. The names of the variables reflect which registers must be used for the MIPS assembly. Do not assume any initial values for the registers. You may use additional registers.

```
if ($t0 == 0) {
  $t1 = 5;
}
```

12.) Convert the following C-like code into MIPS assembly. The names of the variables reflect which registers must be used for the MIPS assembly. Do not assume any initial values for the registers. You may use additional registers.

```
if ($t0 < 5) {
  $t1 = 0;
} else {
  $t1 = 1;
}
```

13.) Convert the following C-like code into MIPS assembly. The names of the variables reflect which registers must be used for the MIPS assembly. Do not assume any initial values for the registers. You may use additional registers.

```
if ($t0 == 0 || $t1 == 1) {
  $t2 = 5;
} else {
  $t2 = 6;
}
```

14.) Convert the following C-like code into MIPS assembly. The names of the variables reflect which registers must be used for the MIPS assembly. Do not assume any initial values for the registers. You may use additional registers.

```
if ($t0 >= 0 && $t0 < $t1) {
  $t2 = 9;
} else {
  $t2 = 0;
}
```

15.) Write a MIPS program that will read integers from the user until 0 is input. Once 0 is input, the program should print the sum of all the numbers read in. As a hint, you should track a running sum, instead of trying to store all the numbers the user read in. If the user immediately inputs a 0, then the running sum should be 0.

16.) Convert the following C-like code into MIPS assembly. The names of the variables reflect which registers must be used for the MIPS assembly. Do not assume any initial values for the registers. You may use additional registers.

```
$t0 = 10
$t1 = 1
$t2 = 0
while ($t1 <= $t0) {
  $t2 = $t2 + $t1;
  $t1++;
}
```

17.) Write a MIPS program that will read in an integer, and will print one of two things:

- Bit 2 is set
- Bit 2 is not set

...depending on whether or not bit 2 of the input number is set. To be clear, bit 0 refers to the rightmost bit in the number.

18.) Convert the following C-like code into MIPS assembly. The names of the variables reflect which registers must be used for the MIPS assembly. Do not assume any initial values for the registers. You may use additional registers.

```
int s0 = 82;
int s1 = s0 << 2;
int s2 = s1 * 20;
int s3 = s2 + 7;
int s4 = s3 - 24;
int s5 = s4 / 3;
```

19.) Convert the following C-like code into MIPS assembly. The names of the variables reflect which registers must be used for the MIPS assembly. Do not assume any initial values for the registers. You may use additional registers. The portions in <<>> will require you to use QtSpim functionality. You do not need to exit the program properly.

```
int s0 = <<read integer from the user>>;
int s1 = 2;
if (s0 < 7) {
  s1 = 3;
}
<<print integer s1>>
```

20.) Convert the following C-like code into MIPS assembly. The names of the variables reflect which registers must be used for the MIPS assembly. Do not assume any initial values for the registers. You may use additional registers. The portions in <<>> will require you to use QtSpim functionality. You do not need to exit the program properly.

```
int s0 = <<read integer from the user>>;
int s1 = 2;
if (s0 < 7) {
  s1 = 3;
} else {
  s1 = s0 + s0;
}
<<print integer s1>>
```

21.) Convert the following C-like code into MIPS assembly. The names of the variables reflect which registers must be used for the MIPS assembly. Do not assume any initial values for the registers. You may use additional registers. The portions in <<>> will require you to use QtSpim functionality. You do not need to exit the program properly.

```
int s0;
int s1 = 1;
for (s0 = 0; s0 < 10; s0++) {
  s1 = s1 * s0;
}
```