I.16: What kind of loop is better: Count up from zero or count down to zero? Why?

I believe the answer comes down to a tradeoff, with both methods having pros and cons.

Count Down

- Many processors have an entire assembly instruction to handle comparisons to zero (branch if equal to zero, for example).
- In contrast, a loop counting up will compare against a constant value, meaning the MCU may have to do lots of data shifting with general purpose registers.
- By counting down, you are constantly comparing your index to zero and taking advantage of the fast instruction. This means, you may be able to reduce 10 clock cycles down to 1.
- The tradeoff here is that your code may be slightly less readable, and, if you're not careful (like I haven't been in the past) you can get nasty bugs like off-by-one errors.

Count Up

- Your binary may run slower for the reasons mentioned above, but your code will be more intuitive and cleaner.

I.5: Why do we use the keyword volatile? (Also R.8)

Example code:

```
int g_interrupt_flag = 0;

if(g_interrupt_flag == 5)
{
  //Do something
}
```

In the code above the variable *g_interrupt_flag* is set to 0 and never touched again in the eyes of the compiler. “Oh”, it says, “I’ll be clever and take that puppy out to save you some code space/latency.” Little does it know that the variable is actually tied to an interrupt and changes when the user pushes a button.

Volatile can be used to tell the compiler that a specific variable will change outside of its scope. Because of this, we’re essentially telling it that we’re doing something funky with this variable that it can’t see/understand. We want it to keep its grubby optimization paws off our variable, even if it means added code space and seemingly wasted speed.