

--- Day 23: Safe Cracking ---

This is one of the top floors of the nicest tower in EBHQ. The Easter Bunny's private office is here, complete with a safe hidden behind a painting, and who wouldn't hide a star in a safe behind a painting?

The safe has a digital screen and keypad for code entry. A sticky note attached to the safe has a password hint on it: "eggs". The painting is of a large rabbit coloring some eggs. You see `7`.

When you go to type the code, though, nothing appears on the display; instead, the keypad comes apart in your hands, apparently having been smashed. Behind it is some kind of socket - one that matches a connector in your `prototype computer`! You pull apart the smashed keypad and extract the logic circuit, plug it into your computer, and plug your computer into the safe.

Now, you just need to figure out what output the keypad would have sent to the safe. You extract the `assembunny code` from the logic chip (your puzzle input).

The code looks like it uses `almost` the same architecture and instruction set that the `monorail computer` used! You should be able to `use the same assembunny interpreter` for this as you did there, but with one new instruction:

`tgl x` toggles the instruction `x` away (pointing at instructions like `in兹` does: positive means forward; negative means backward):

- For `one-argument` instructions, `inc` becomes `dec`, and all other one-argument instructions become `inc`.
- For `two-argument` instructions, `in兹` becomes `cpy`, and all other two-argument instructions become `in兹`.
- The arguments of a toggled instruction are `not affected`.
- If an attempt is made to toggle an instruction outside the program, `nothing happens`.
- If toggling produces an `invalid instruction` (like `cpy 1 2`) and an attempt is later made to execute that instruction, skip it instead.
- If `tgl` toggles itself (for example, if `a` is `0`, `tgl a` would target itself and become `inc a`), the resulting instruction is not executed until the next time it is reached.

For example, given this program:


```
cpy 2 a
tgl a
tgl a
tgl a
cpy 1 a
dec a
dec a
```

- `cpy 2 a` initializes register `a` to `2`.
- The first `tgl a` toggles an instruction `a` (`2`) away from it, which changes the third `tgl a` into `inc a`.
- The second `tgl a` also modifies an instruction `2` away from it, which changes the `cpy 1 a` into `in兹 1 a`.
- The fourth line, which is now `inc a`, increments `a` to `3`.
- Finally, the fifth line, which is now `in兹 1 a`, jumps `a` (`3`) instructions ahead, skipping the `dec a` instructions.

In this example, the final value in register `a` is `3`.

Our `sponsors` help make Advent of Code possible:


`eSpark Learning` - Solve the greatest puzzle of our day - transform education



Stand out with a website

Everything you need to stand out online. Create a beautiful website with Squarespace.

Squarespace



The rest of the electronics seem to place the keypad entry (the number of eggs, `7`) in register `a`, run the code, and then send the value left in register `a` to the safe.

What value should be sent to the safe?

Your puzzle answer was `12703`.

--- Part Two ---

The safe doesn't open, but it **does** make several angry noises to express its frustration.

You're quite sure your logic is working correctly, so the only other thing is... you check the painting again. As it turns out, colored eggs are still eggs. Now you count `12`.

As you run the program with this new input, the prototype computer begins to **overheat**. You wonder what's taking so long, and whether the lack of any instruction more powerful than "add one" has anything to do with it. Don't bunnies usually multiply?

Anyway, what value should actually be sent to the safe?

Your puzzle answer was `479009263`.

Both parts of this puzzle are complete! They provide two gold stars: **

At this point, you should **return to your advent calendar** and try another puzzle.

If you still want to see it, you can **get your puzzle input**.

You can also **[Share]** this puzzle.