--- Day 21: Chronal Conversion ---

You should have been watching where you were going, because as you wander the new North Pole base, you trip and fall into a very deep hole!

Just kidding. You're falling through time again.

If you keep up your current pace, you should have resolved all of the temporal anomalies by the next time the device activates. Since you have very little interest in browsing history in 500-year increments for the rest of your life, you need to find a way to get back to your present time.

After a little research, you discover two important facts about the behavior of the device:

First, you discover that the device is hard-wired to always send you back in time in 500-year increments. Changing this is probably not feasible.

Second, you discover the activation system (your puzzle input) for the time travel module. Currently, it appears to run forever without halting.

If you can cause the activation system to halt at a specific moment, maybe you can make the device send you so far back in time that you cause an integer underflow in time itself and wrap around back to your current time!

The device executes the program as specified in manual section one and manual section two.

Your goal is to figure out how the program works and cause it to halt. You can only control register `0`; every other register begins at `0` as usual.

Because time travel is a dangerous activity, the activation system begins with a few instructions which verify that bitwise AND (via `bani`) does a numeric operation and not an operation as if the inputs were interpreted as strings. If the test fails, it enters an infinite loop re-running the test instead of allowing the program to execute normally. If the test passes, the program continues, and assumes that all other bitwise operations (`banr`, `bori`, and `borr`) also interpret their inputs as numbers. (Clearly, the Elves who wrote this system were worried that someone might introduce a bug while trying to emulate this system with a scripting language.)

What is the lowest non-negative integer value for register `0` that causes the program to halt after executing the fewest instructions? (Executing the same instruction multiple times counts as multiple instructions executed.)

Your puzzle answer was `15615244`.

--- Part Two ---

In order to determine the timing window for your underflow exploit, you also need an upper bound:

What is the lowest non-negative integer value for register `0` that causes the program to halt after executing the most instructions? (The program must actually halt; running forever does not count as halting.)

Your puzzle answer was `12963935`.

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.