--- Day 5: Sunny with a Chance of Asteroids ---

You're starting to sweat as the ship makes its way toward Mercury. The Elves suggest that you get the air conditioner working by upgrading your ship computer to support the Thermal Environment Supervision Terminal.

The Thermal Environment Supervision Terminal (TEST) starts by running a **diagnostic program** (your puzzle input). The TEST diagnostic program will run on your existing Intcode computer after a few modifications:

First, you'll need to add **two new instructions**:

- Opcode `3` takes a single integer as **input** and saves it to the position given by its only parameter. For example, the instruction `3,50` would take an input value and store it at address `50`.
- Opcode `4` **outputs** the value of its only parameter. For example, the instruction `4,50` would output the value at address `50`.

Programs that use these instructions will come with documentation that explains what should be connected to the input and output. The program `3,0,4,0,99` outputs whatever it gets as input, then halts.

Second, you'll need to add support for **parameter modes**:

Each parameter of an instruction is handled based on its parameter mode. Right now, your ship computer already understands parameter mode `0`, **position mode**, which causes the parameter to be interpreted as a **position** - if the parameter is `50`, its value is **the value stored at address `50` in memory**. Until now, all parameters have been in position mode.

Now, your ship computer will also need to handle parameters in mode `1`, **immediate mode**. In immediate mode, a parameter is interpreted as a **value** - if the parameter is `50`, its value is simply `50`.

Parameter modes are stored in the same value as the instruction's opcode. The opcode is a two-digit number based only on the ones and tens digit of the value, that is, the opcode is the rightmost two digits of the first value in an instruction. Parameter modes are single digits, one per parameter, read right-to-left from the opcode: the first parameter's mode is in the hundreds digit, the second parameter's mode is in the thousands digit, the third parameter's mode is in the ten-thousands digit, and so on. Any missing modes are `0`.

For example, consider the program `1002,4,3,4,33`.

The first instruction, `1002,4,3,4`, is a **multiply** instruction - the rightmost two digits of the first value, `02`, indicate opcode `2`, multiplication. Then, going right to left, the parameter modes are `0` (hundreds digit), `1` (thousands digit), and `0` (ten-thousands digit, not present and therefore zero):

```
ABCDE
 1002

DE - two-digit opcode,      02 == opcode 2
 C - mode of 1st parameter,  0 == position mode
 B - mode of 2nd parameter,  1 == immediate mode
 A - mode of 3rd parameter,  0 == position mode,
                                  omitted due to being a leading zero
```

This instruction multiplies its first two parameters. The first parameter, `4` in position mode, works like it did before - its value is the value

stored at address `4` (`33`). The second parameter, `3` in immediate mode, simply has value `3`. The result of this operation, `33 * 3 = 99`, is written according to the third parameter, `4` in position mode, which also works like it did before - `99` is written to address `4`.

Parameters that an instruction writes to will **never be in immediate mode**.

Finally, some notes:

- It is important to remember that the instruction pointer should increase by **the number of values in the instruction** after the instruction finishes. Because of the new instructions, this amount is no longer always `4`.
- Integers can be negative: `1101,100,-1,4,0` is a valid program (find `100 + -1`, store the result in position `4`).

The TEST diagnostic program will start by requesting from the user the ID of the system to test by running an **input** instruction - provide it `1`, the ID for the ship's air conditioner unit.

It will then perform a series of diagnostic tests confirming that various parts of the Intcode computer, like parameter modes, function correctly. For each test, it will run an **output** instruction indicating how far the result of the test was from the expected value, where `0` means the test was successful. Non-zero outputs mean that a function is not working correctly; check the instructions that were run before the output instruction to see which one failed.

Finally, the program will output a **diagnostic code** and immediately halt. This final output isn't an error; an output followed immediately by a halt means the program finished. If all outputs were zero except the diagnostic code, the diagnostic program ran successfully.

After providing `1` to the only input instruction and passing all the tests, **what diagnostic code does the program produce?**

Your puzzle answer was `5074395`.

--- Part Two ---

The air conditioner comes online! Its cold air feels good for a while, but then the TEST alarms start to go off. Since the air conditioner can't vent its heat anywhere but back into the spacecraft, it's actually making the air inside the ship **warmer**.

Instead, you'll need to use the TEST to extend the thermal radiators. Fortunately, the diagnostic program (your puzzle input) is already equipped for this. Unfortunately, your Intcode computer is not.

Your computer is only missing a few opcodes:

- Opcode `5` is **jump-if-true**: if the first parameter is **non-zero**, it sets the instruction pointer to the value from the second parameter. Otherwise, it does nothing.
- Opcode `6` is **jump-if-false**: if the first parameter **is zero**, it sets the instruction pointer to the value from the second parameter. Otherwise, it does nothing.
- Opcode `7` is **less than**: if the first parameter is **less than** the second parameter, it stores `1` in the position given by the third parameter. Otherwise, it stores `0`.
- Opcode `8` is **equals**: if the first parameter is **equal to** the second parameter, it stores `1` in the position given by the third parameter. Otherwise, it stores `0`.

Like all instructions, these instructions need to support **parameter modes** as described above.

Normally, after an instruction is finished, the instruction pointer increases by the number of values in that instruction. **However**, if the instruction modifies the instruction pointer, that value is used and the instruction pointer is **not automatically increased**.

For example, here are several programs that take one input, compare it to the value `8`, and then produce one output:

- `3,9,8,9,10,9,4,9,99,-1,8` - Using **position mode**, consider whether the input is **equal to** `8`; output `1` (if it is) or `0` (if it is not).
- `3,9,7,9,10,9,4,9,99,-1,8` - Using **position mode**, consider whether the input is **less than** `8`; output `1` (if it is) or `0` (if it is not).
- `3,3,1108,-1,8,3,4,3,99` - Using **immediate mode**, consider whether the input is **equal to** `8`; output `1` (if it is) or `0` (if it is not).
- `3,3,1107,-1,8,3,4,3,99` - Using **immediate mode**, consider whether the input is **less than** `8`; output `1` (if it is) or `0` (if it is not).

Here are some jump tests that take an input, then output `0` if the input was zero or `1` if the input was non-zero:

- `3,12,6,12,15,1,13,14,13,4,13,99,-1,0,1,9` (using **position mode**)
- `3,3,1105,-1,9,1101,0,0,12,4,12,99,1` (using **immediate mode**)

Here's a larger example:

```
3,21,1008,21,8,20,1005,20,22,107,8,21,20,1006,20,31,
1106,0,36,98,0,0,1002,21,125,20,4,20,1105,1,46,104,
999,1105,1,46,1101,1000,1,20,4,20,1105,1,46,98,99
```

The above example program uses an input instruction to ask for a single number. The program will then output `999` if the input value is below `8`, output `1000` if the input value is equal to `8`, or output `1001` if the input value is greater than `8`.

This time, when the TEST diagnostic program runs its input instruction to get the ID of the system to test, **provide it** `5`, the ID for the ship's thermal radiator controller. This diagnostic test suite only outputs one number, the **diagnostic code**.

What is the diagnostic code for system ID `5`**?**

Your puzzle answer was `8346937`.

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.