--- Day 14: Docking Data ---

As your ferry approaches the sea port, the captain asks for your help again. The computer system that runs this port isn't compatible with the docking program on the ferry, so the docking parameters aren't being correctly initialized in the docking program's memory.

After a brief inspection, you discover that the sea port's computer system uses a strange bitmask system in its initialization program. Although you don't have the correct decoder chip handy, you can emulate it in software!

The initialization program (your puzzle input) can either update the bitmask or write a value to memory. Values and memory addresses are both 36-bit unsigned integers. For example, ignoring bitmasks for a moment, a line like `mem[8] = 11` would write the value `11` to memory address `8`.

The bitmask is always given as a string of 36 bits, written with the most significant bit (representing `2^35`) on the left and the least significant bit (`2^0`, that is, the `1`s bit) on the right. The current bitmask is applied to values immediately before they are written to memory: a `0` or `1` overwrites the corresponding bit in the value, while an `X` leaves the bit in the value unchanged.

For example, consider the following program:

```
mask = XXXXXXXXXXXXXXXXXXXXXXXXXXXXX1XXXX0X
mem[8] = 11
mem[7] = 101
mem[8] = 0
```

This program starts by specifying a bitmask (`mask = ....`). The mask it specifies will overwrite two bits in every written value: the `2`s bit is overwritten with `0`, and the `64`s bit is overwritten with `1`.

The program then attempts to write the value `11` to memory address `8`. By expanding everything out to individual bits, the mask is applied as follows:

```
value:  000000000000000000000000000000001011  (decimal 11)
mask:   XXXXXXXXXXXXXXXXXXXXXXXXXXXXX1XXXX0X
result: 000000000000000000000000000001001001  (decimal 73)
```

So, because of the mask, the value `73` is written to memory address `8` instead. Then, the program tries to write `101` to address `7`:

```
value:  000000000000000000000000000001100101  (decimal 101)
mask:   XXXXXXXXXXXXXXXXXXXXXXXXXXXXX1XXXX0X
result: 000000000000000000000000000001100101  (decimal 101)
```

This time, the mask has no effect, as the bits it overwrote were already the values the mask tried to set. Finally, the program tries to write `0` to address `8`:

```
value:  000000000000000000000000000000000000  (decimal 0)
mask:   XXXXXXXXXXXXXXXXXXXXXXXXXXXXX1XXXX0X
result: 000000000000000000000000000001000000  (decimal 64)
```

`64` is written to address `8` instead, overwriting the value that was there previously.

To initialize your ferry's docking program, you need the sum of all values left in memory after the initialization program completes. (The entire 36-

bit address space begins initialized to the value `0` at every address.) In the above example, only two values in memory are not zero - `101` (at address `7`) and `64` (at address `8`) - producing a sum of `165`.

Execute the initialization program. What is the sum of all values left in memory after it completes?

To begin, get your puzzle input.

Answer: [Submit]

You can also [Share] this puzzle.