day14

December 15, 2020

1 Day 14 Bitmasks

Bit masks are fun things, especially in this case.

We have a fairly simple concept here, create a sparse memory array, and then go through the instructions, writing to memory every instruction, and passing the memory blob through a bitmask modification function.

Python has some nice tools for bit arrays, so this shouldn't be too hard, but the modification needs to take the number in decimal, turn it into a bit array, and then go through the bit array replacing bits as needed. A ternary array would be more useful than a binary array here, but that would be horribly complex to implement, so instead we'll turn the bit array into a sequence of bits to toggle, and apply that to the binary form of the value to write.

[14]:
```
import ipytest
ipytest.autoconfig()
import bitstring

def create_converter(bitarray):
    values = {}
    for i,b in enumerate(bitarray[::-1]):
        if b != "X":
            values[i] = int(b)
    return values

def convert(value, converter):
    bits = bitstring.BitArray(uint=value,length=36)
    for i,b in converter.items():
        bits[-i-1] = b
    return bits.uint

test_converter = create_converter("XXXXXXXXXXXXXXXXXXXXXXXXXXXXX1XXXX0X")
assert 73 == convert(11, test_converter)
assert 101 == convert(101, test_converter)
assert 64 == convert(0, test_converter)
```

Grand, that seems to work sensibly.

Now looking at the data, it looks like we'll have to parse the instructions, and that there will be lots of mask= calls to change the mask, so I think we'll just step through line by line and then set

the memory, and validate that it did the right thing

[19]:
```
import re
mem_re = re.compile("mem\[(\d+)\] = (\d+)")
def process_lines(lines):
    mem = {}
    converter = None
    for line in lines:
        if line.startswith("mask = "):
            converter = create_converter(line[7:])
        if line.startswith("mem"):
            nums = [int(x) for x in mem_re.match(line).groups()]
            mem[nums[0]]=convert(nums[1], converter)
    return mem

test_lines="""mask = XXXXXXXXXXXXXXXXXXXXXXXXXXXXX1XXXX0X
mem[8] = 11
mem[7] = 101
mem[8] = 0""".split("\n")
test_mem = process_lines(test_lines)
assert test_mem[8] == 64
assert test_mem[7] == 101
assert sum(test_mem.values()) == 165
```

That works better than I thought.... Let's see if it works on real data

[20]:
```
mem = process_lines([line.strip() for line in open("day14.txt")])
print(sum(mem.values()))
```

```
17028179706934
```