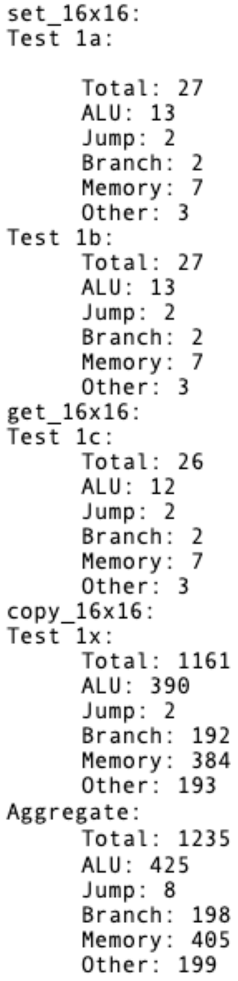
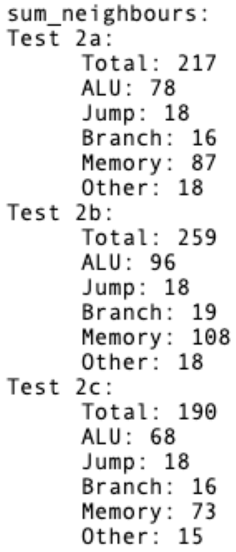
Part 1



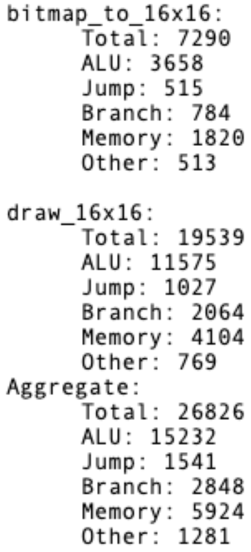
```
set_16x16:
Test 1a:

     Total: 27
     ALU: 13
     Jump: 2
     Branch: 2
     Memory: 7
     Other: 3
Test 1b:
     Total: 27
     ALU: 13
     Jump: 2
     Branch: 2
     Memory: 7
     Other: 3
get_16x16:
Test 1c:
     Total: 26
     ALU: 12
     Jump: 2
     Branch: 2
     Memory: 7
     Other: 3
copy_16x16:
Test 1x:
     Total: 1161
     ALU: 390
     Jump: 2
     Branch: 192
     Memory: 384
     Other: 193
Aggregate:
     Total: 1235
     ALU: 425
     Jump: 8
     Branch: 198
     Memory: 405
     Other: 199
```

Part 2



```
sum_neighbours:
Test 2a:
     Total: 217
     ALU: 78
     Jump: 18
     Branch: 16
     Memory: 87
     Other: 18
Test 2b:
     Total: 259
     ALU: 96
     Jump: 18
     Branch: 19
     Memory: 108
     Other: 18
Test 2c:
     Total: 190
     ALU: 68
     Jump: 18
     Branch: 16
     Memory: 73
     Other: 15
```

Part 3



```
bitmap_to_16x16:
     Total: 7290
     ALU: 3658
     Jump: 515
     Branch: 784
     Memory: 1820
     Other: 513

draw_16x16:
     Total: 19539
     ALU: 11575
     Jump: 1027
     Branch: 2064
     Memory: 4104
     Other: 769
Aggregate:
     Total: 26826
     ALU: 15232
     Jump: 1541
     Branch: 2848
     Memory: 5924
     Other: 1281
```

As we can see from the tests above, the number of instructions increase as we move through the program. Getting and setting the 16x16 byte array run about the same amount of instructions where copying has a lot of instructions. What is interesting is that I had anticipated the number of instructions in sum_neighbours to be a little more than 9 times the amount of instructions in get_16x16. This is confirmed (26x9 =234) where the differences in the tests come in is when a byte doesn't exist because the column and row number were out of bounds. We can also see the branching instruction calls start to bump up.

Part 3 is where instruction counts are getting ***real*** big. This is because we're essentially building a new 16x16 byte array from words we're reading into the program. Again we are calling set_16x16, 256(16x16) times, so 7290/265 is around 28 instructions, about the same as set_16x16 in the first test, adding a few more instructions needed to run bitmap_to_16x16. And then we just get ridiculous when we call draw_16x16 as there are so many instructions translating a bit to a picture on a display.

From my findings on these test results, if we want to optimize a program it has to optimize the most called functions that we use a lot. Even if we remove say 3 instructions from set or get_16x16 (lets go with a nice round 24 vs 27), we could make sum_neighbours and bitmap_to_16x16 12% more efficient (216 instructions instead of 243 for sum and 6144 instead of 6912 for bitmap).