Mystery - David Parsons

This directory is holds my implementation of the mystery.s given in the spec that has been converted to C code the has the same functionality as the x86.

Solving Mystery.s

Most of the challenge in understanding mystery.s came from understanding what num was. But to begin I almost immediately guessed that dothething was fibonacci because of the way it was described in the spec. After discovering that 165580141 was in fact the 41st fibonacci number, understanding the assembly became much easier to understand.

More at the time understandings can be found as comments in mystery.c

add

This was pretty straight forward given the name of the function.

dothething (fibonacci)

My approach to figuring out exactly what this function did came from following along the assembly with how I would program fibonacci if I was writing it in assembly. This made it easy to pick up on things like the loops for adding values and why certain class were occurring at certain time. More importantly it made me question why deviates from my logic were occurring. This was most important for discovering the optimization of the calculations using the num array. After googling the line 114, I did some quick math and discovered that num was an array of ints.

main

Understanding this was pretty simple after figuring out the optimization used for calculating fibonacci numbers. As it was as simple as assigning all the value in the array to -1 at the start and the calculating the fibonacci number and then printing the value.

The Optimization Flag

The assembly computed under the optimization flag takes advantage of scaling factors and operand arithmetic, in order to avoid calling timely and unnecessary add, sub, mul, and div operations. This can be done as the compiler is aware of all of the space that will be required for running each function in mystery, this way some local variables can be rendered useless, and calculations can be consolidated into single lines within other operations. Overall these improvements reduce the time and space complexity of the assembly program.