Shammah Thao

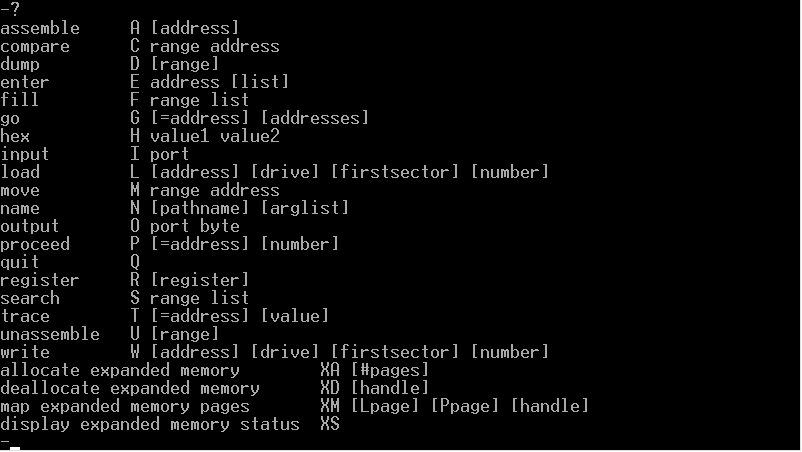
EEE 174 - CpE 185 Lab Section #2

Monday & Wednesday

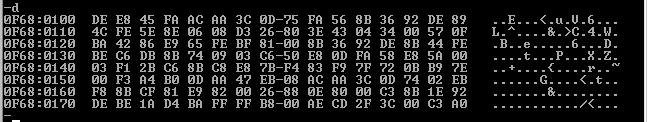
Lab 1: x86 and C refresher

Dahlquist

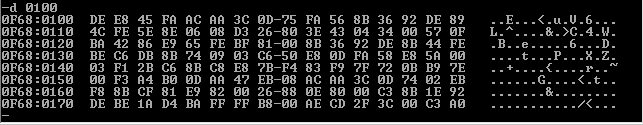
**Part 1: Intro to debug and C refresher**



First thing we had to do was get access to a debugging program in a provided virtual machine. Going into the program called MS-Dos prompt we type “debug” then “?” to open a listing of commands.



Using the command “d” (dump) we display content of the memory location.

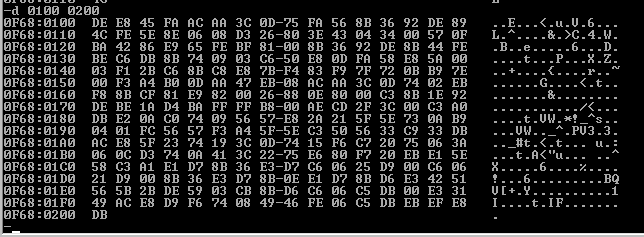


Using d 0100 we focus on displaying listing from the memory location 0100 and so on.

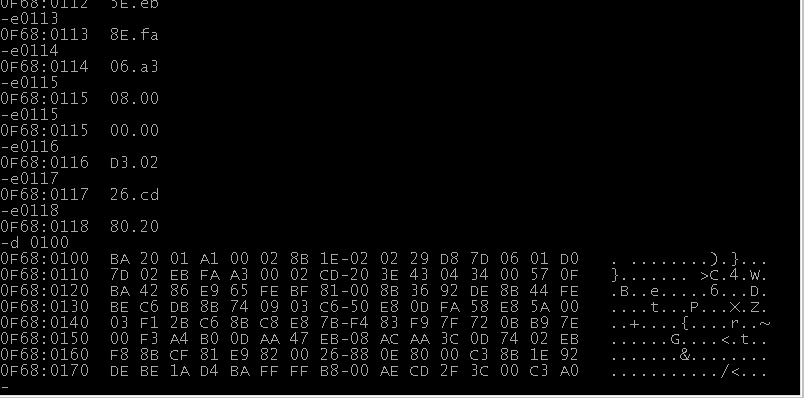


We then used d 0100 0110 to be even more specific on what memory location to view.

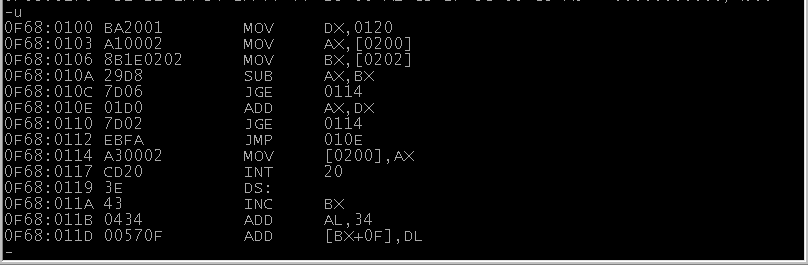
The two listing of number are basically like a between bars. Tells the program that you want to view the memory location between these two number



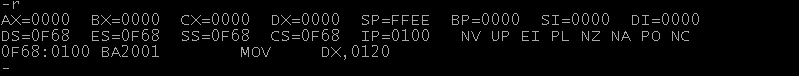
Following up, we used d 0100 0200, to display the number memory between 0100 and 0200.



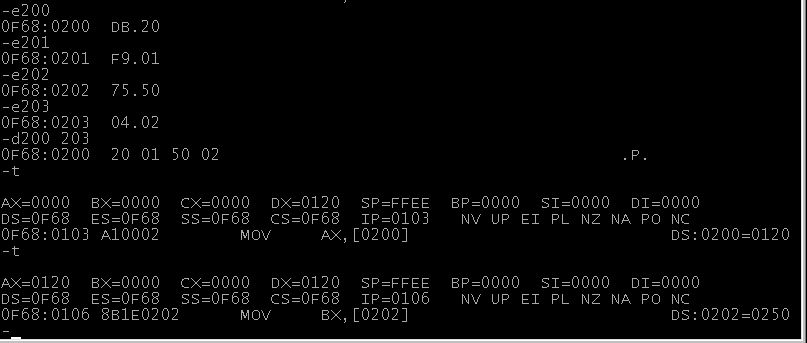
When then used the command “e” (enter) which allowed us to edit the memory location. we used e100 as the starting point then replaced it with the number we were given.

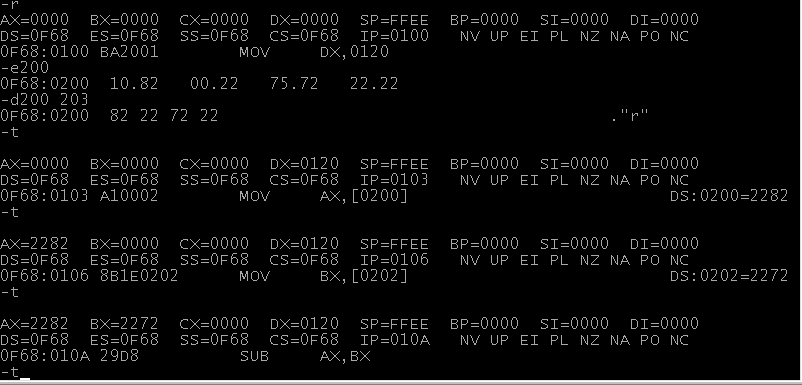


We then used the command “u” (unassembled) to view the changes that was done to the memory location. its tells us what was move to where and where the number were stored.

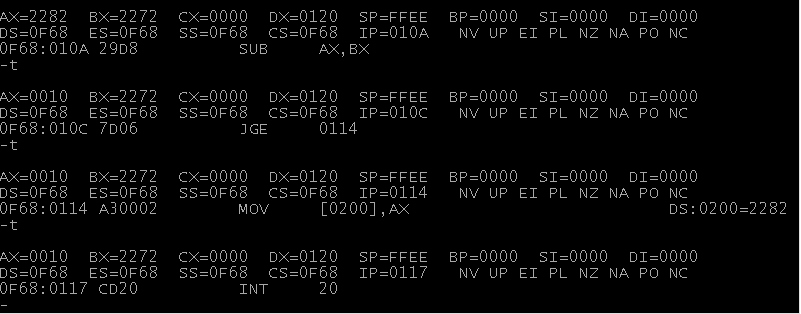


Using “r” (register modify) we set a instruction pointer(IP) at the beginning of the program that we first edited which is 0100.

we then used the “t” (trace), which basically shows that was done to the program set step by step.

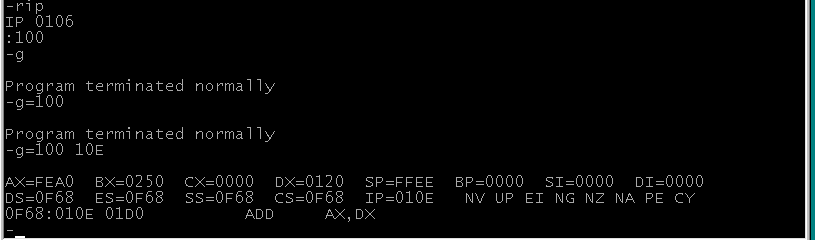


This is one of the trace for the non loop one



Until the ending at int 20



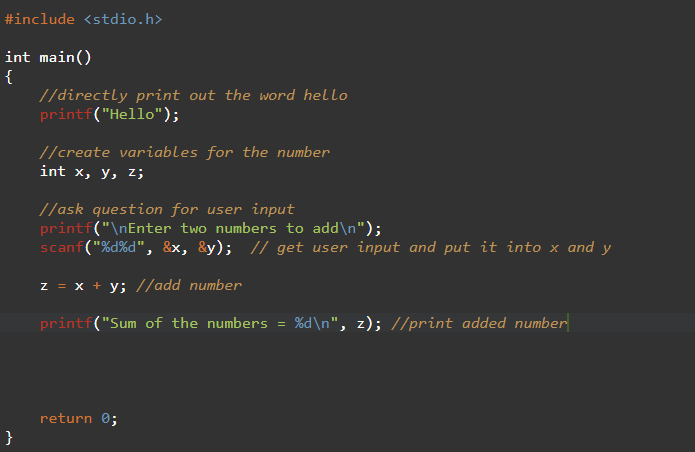


Using “g” (go), we were able to terminate out of the debugger. We also used “g” to break out specific breakpoint. Breaking at different addresses that we put in.

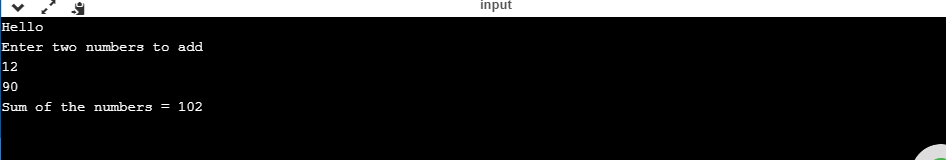
1. The function of the program is to basically tell you what the program is going to do, either its trying to move a memory location, or its trying to add or subtraction a memory value. Until you get to int 20h, which is the end program

**C Refresher**

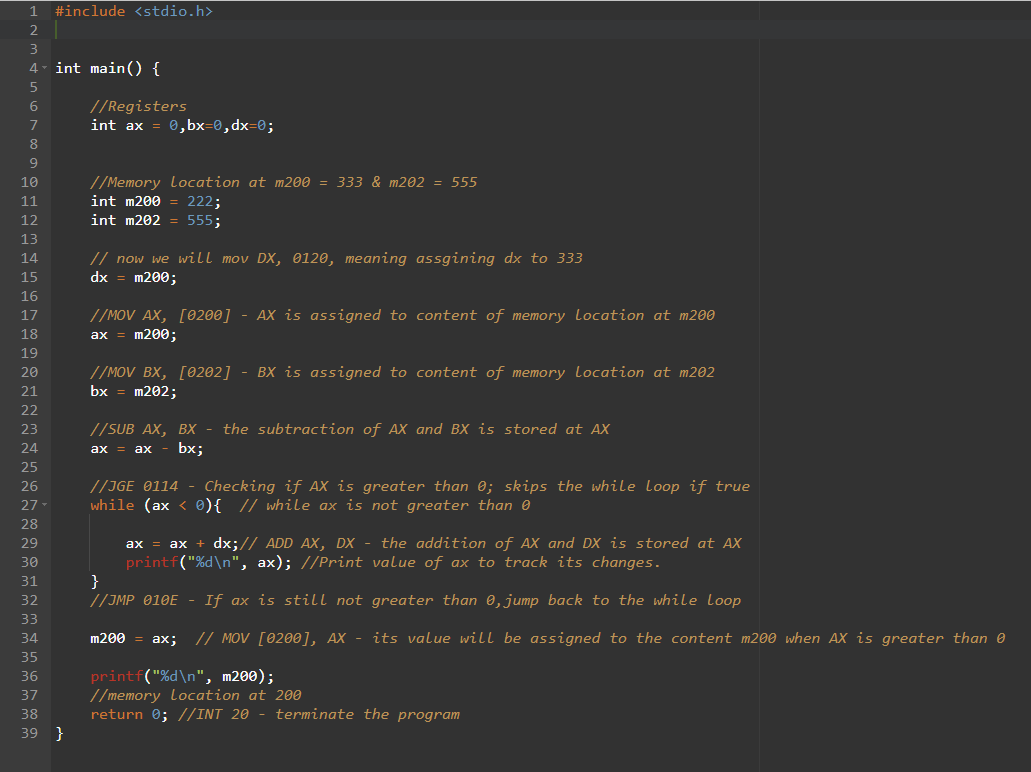
Using [www.onlinegdb.com](http://www.onlinegdb.com) , an online compiler for c,



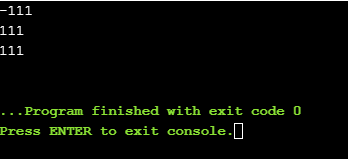
Wrote hello in C program. Also wrong a code to take in user input to add two numbers.



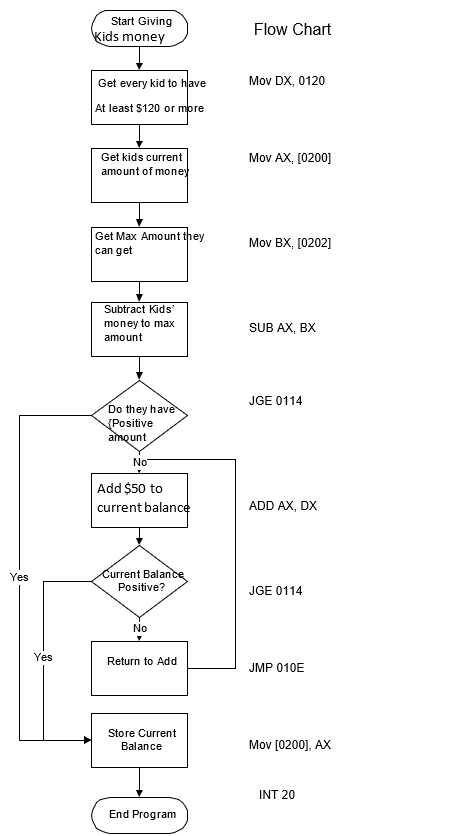
The output shows Hello and two number being entered and added together.



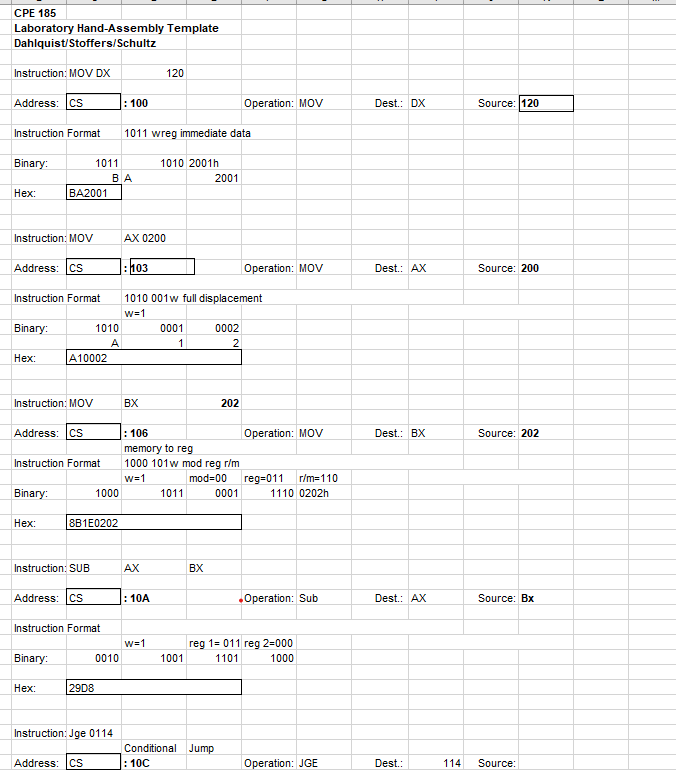
For this code, I rewrote what was happening in the debugger in c language. With comments telling what is happening during each of the process.

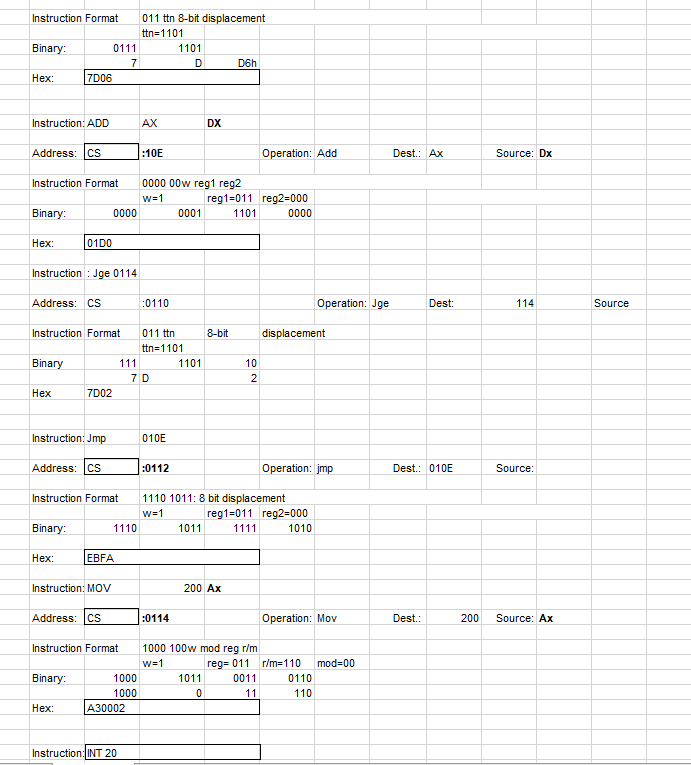


the result from the out shows the it had to go through the loop twice to double check that it is greater than 0. Also showing the final result of m200 at the end.

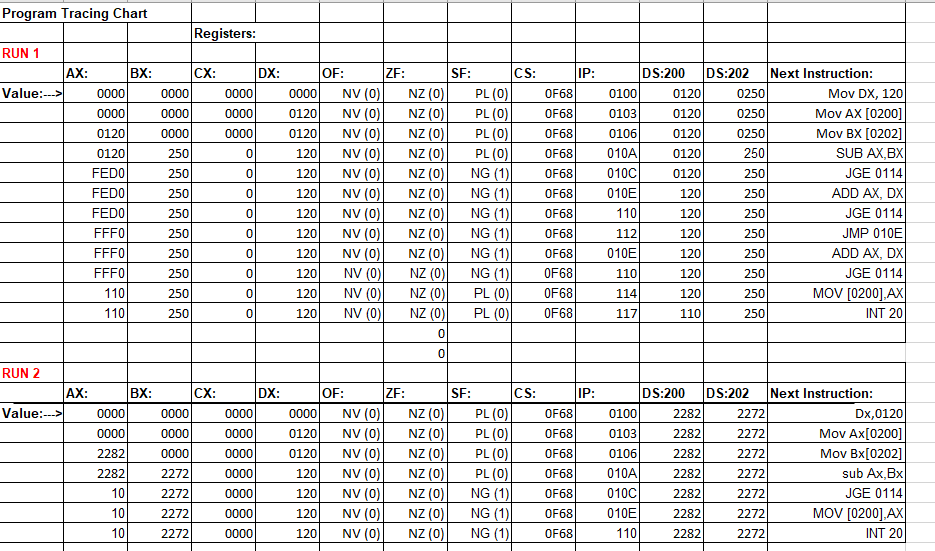
****

Using this flow chart to understand how to the debug work and its functionality with the codes that was input into it





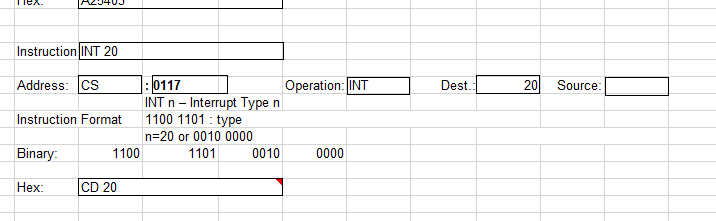
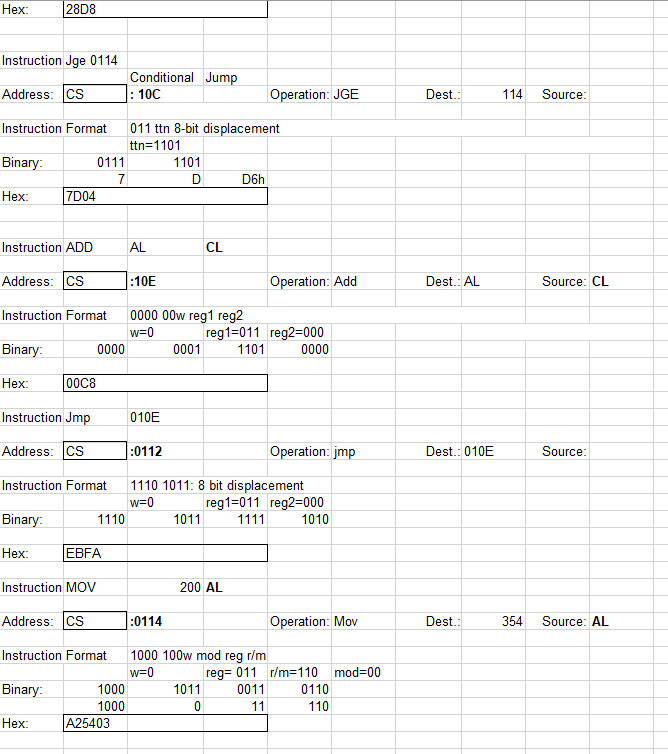
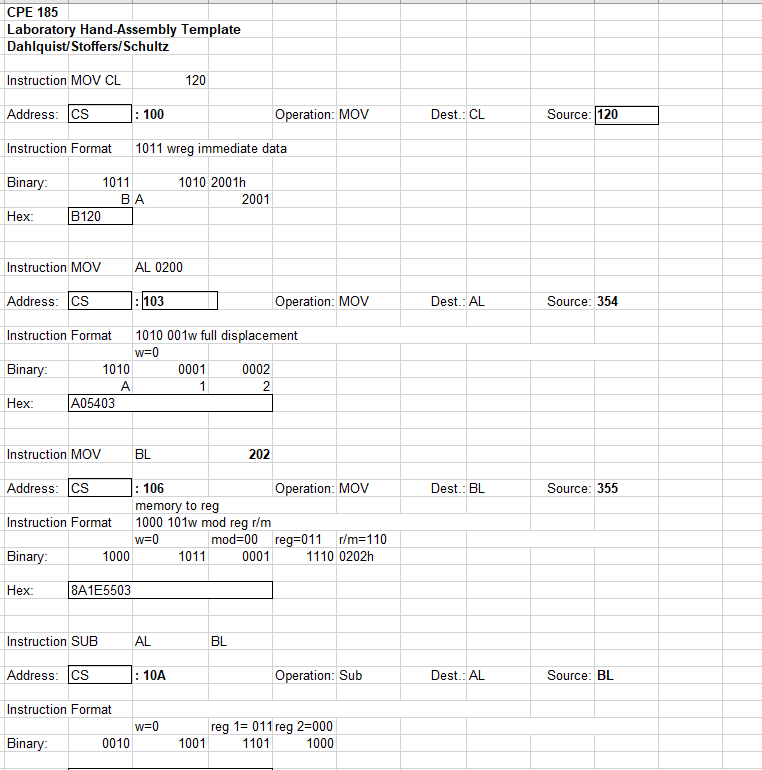
The following tracing chart was used then after first getting the instruction of the debugger. We would then try to get the hex which is the machine language that we would need.



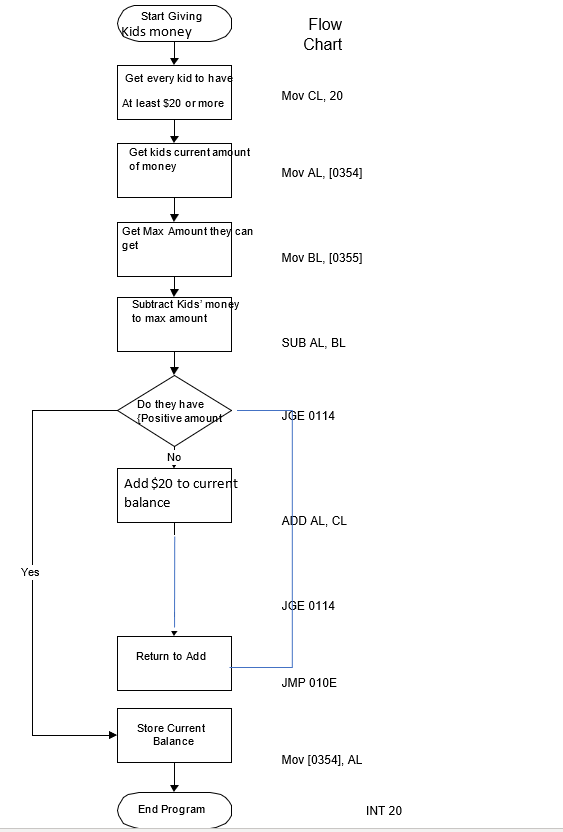
As i went the program, we had to use tracing in the debugger to look step by step on what changed in our register and flags. In the tracing chart, one did a loop and the other just checked if the number was big enough and just skipped through the loop after the first initial check.

**Part2: Hand assembly and C programming**

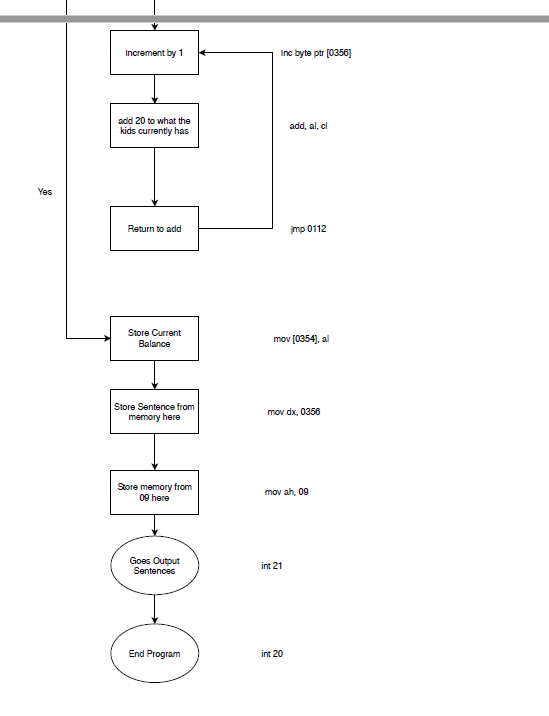
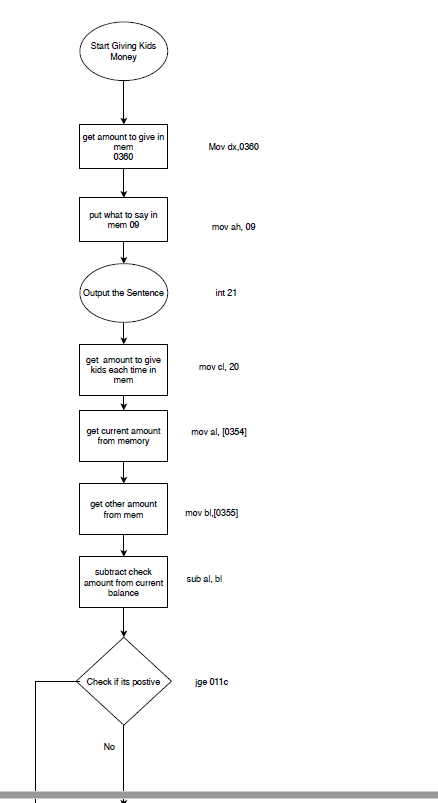
To start, we had to set up a hand assembly, with modified register. We are suppose to use our own unqie register, mine was 354. Using 354, we replaced the original from part 1.



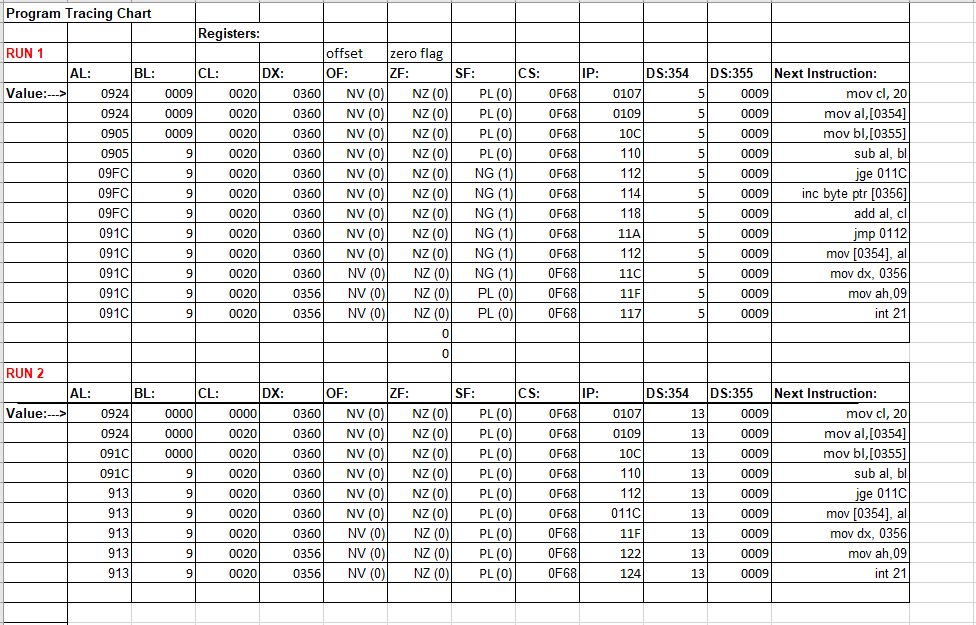
The flow chart below, was made with the same intent as the one from part 1 of this lab. Just modifying the register to match the code. Following the sequence of how the code should be executed.



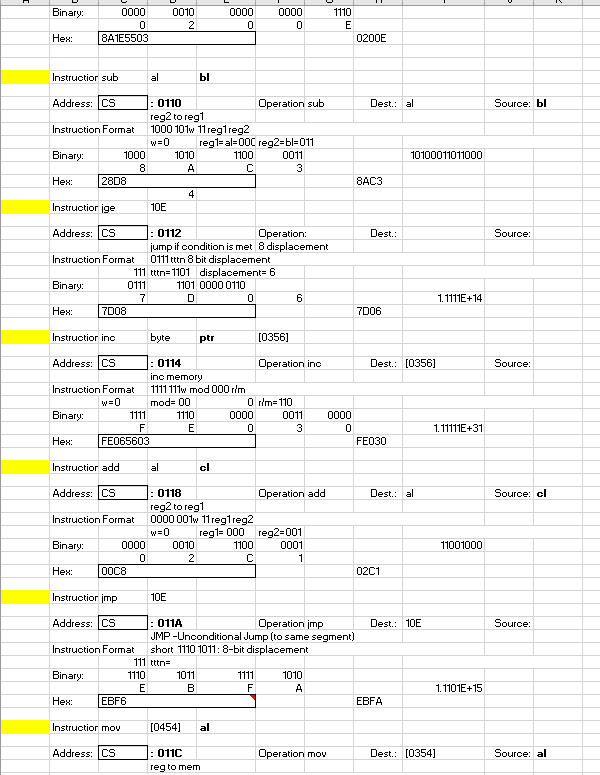
The second flow care is a flow chart that included the increment and creating a title for the debugger.



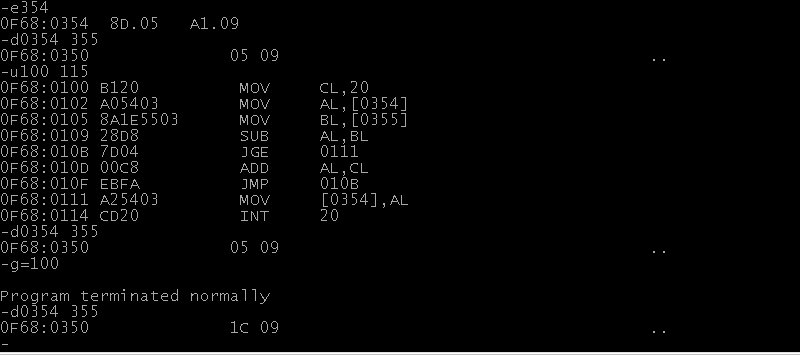
The tracing chart below was used to trace the program after the title and increment was included into the program. We let it ran and over time the sign flag would change then revert back while not being in the loop



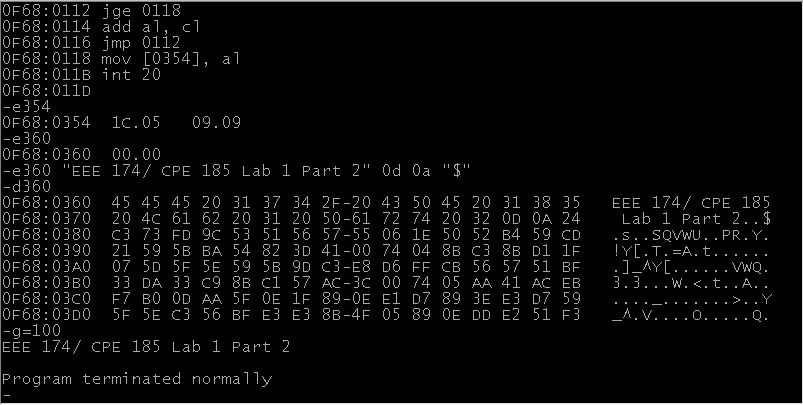
The hand assemebly was made to include the loop and the incrementing to get the machine code needed for the demo.



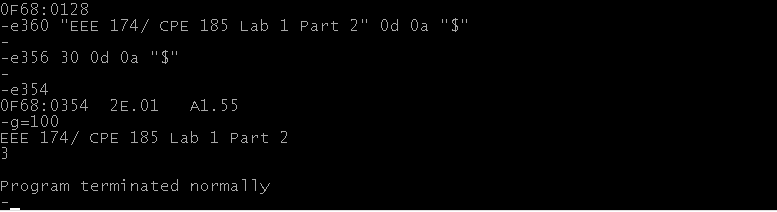
Following through the steps first part of the lab instruction, we were able to make it excute with g=100 without any issue. The last output shows that the original number 5 have changed to 1C due to the reg going through the loop.



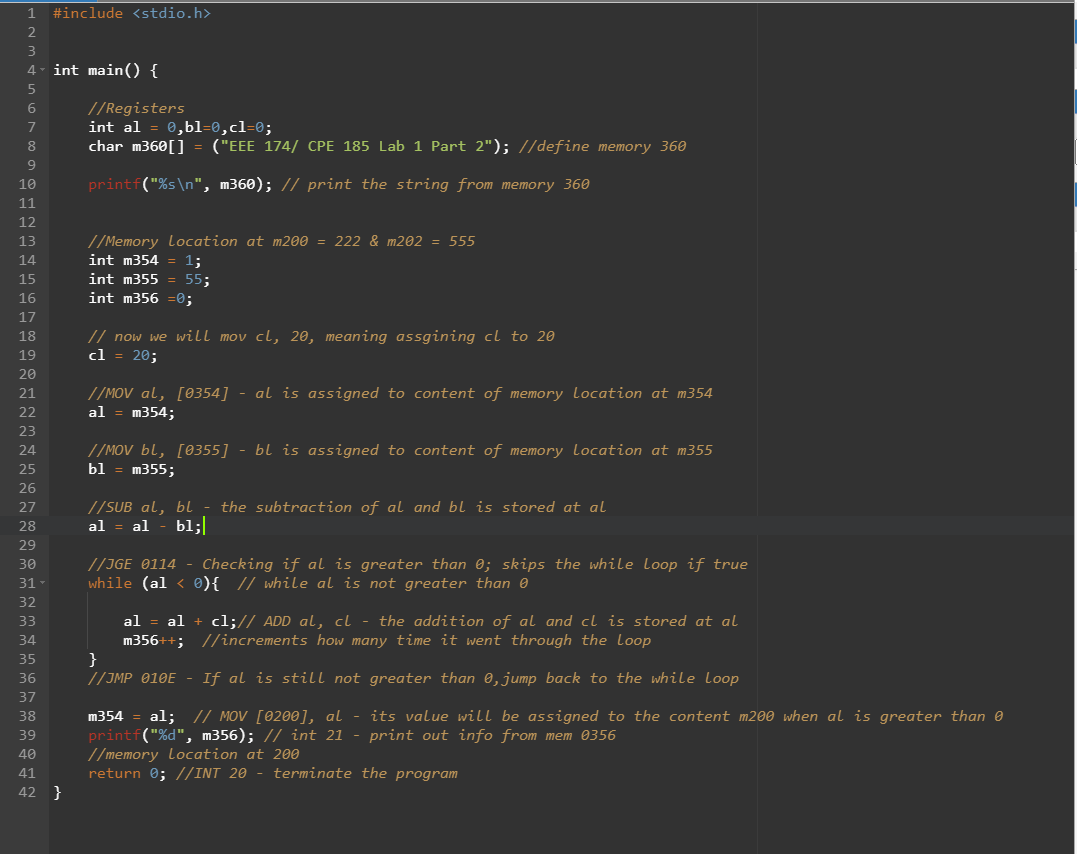
Next on the instruction, we had it so it would be about to output a title from the debugger. Inputting in the instruction and the quote into the registers, we were about to output a title for our code.

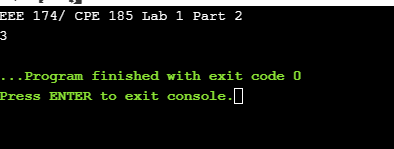


For the final part, we included the incrementing and title into the debugger. We were about to get it to increase. I used the number 1 and 55 in the register which made it looped 3x



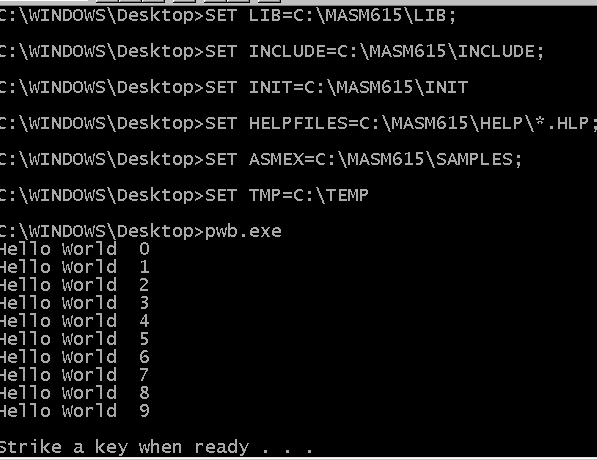
For the C coding that involves, putting in the incrementing and the title, we just had to make a variable for the string and create a variable to count the number of increment in the loop.



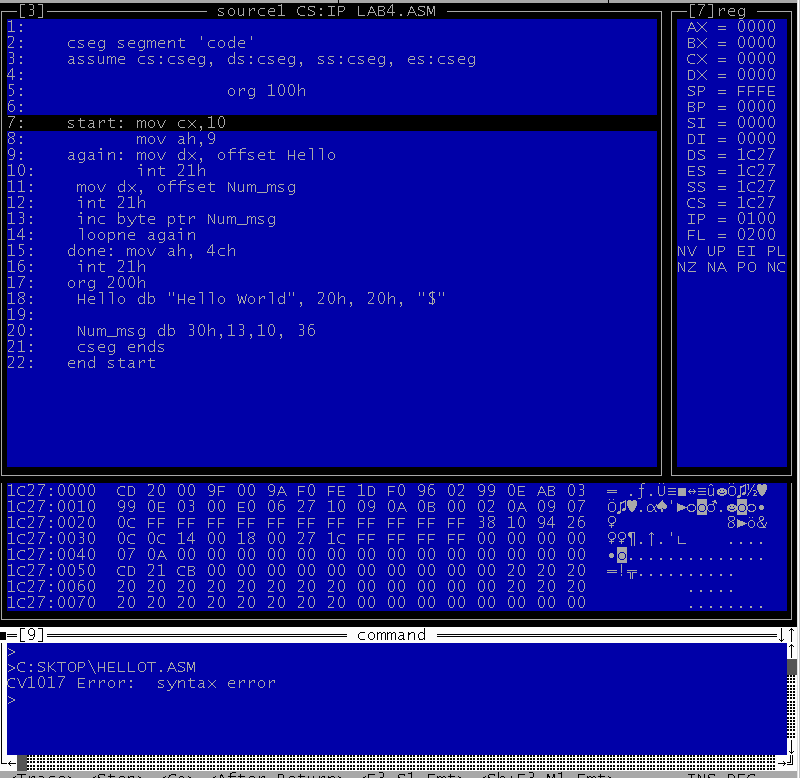


**Part 3: Microsoft’s Assembly language Development System (MASM)**

For the beginning part of the lab, we about to set up the pwd’s editor using the given instruction. We then set the given code into an asm file which ran the Following output.

****

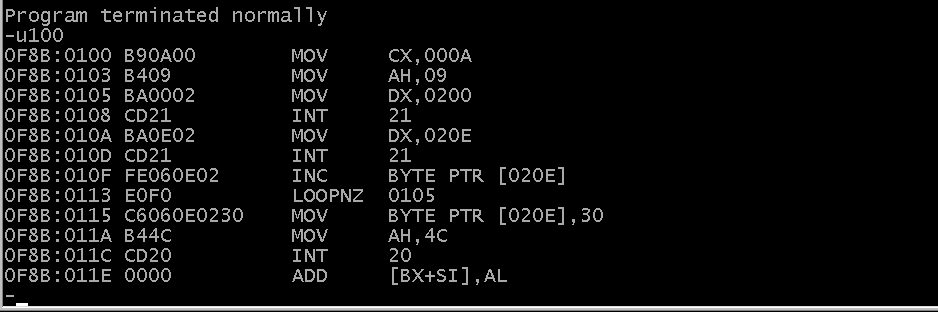
Testing out the debugger, one of the feature of the pwd, we ever able to see step by step of what was happening during the code ran, by pressing F10 we were able to step through it.



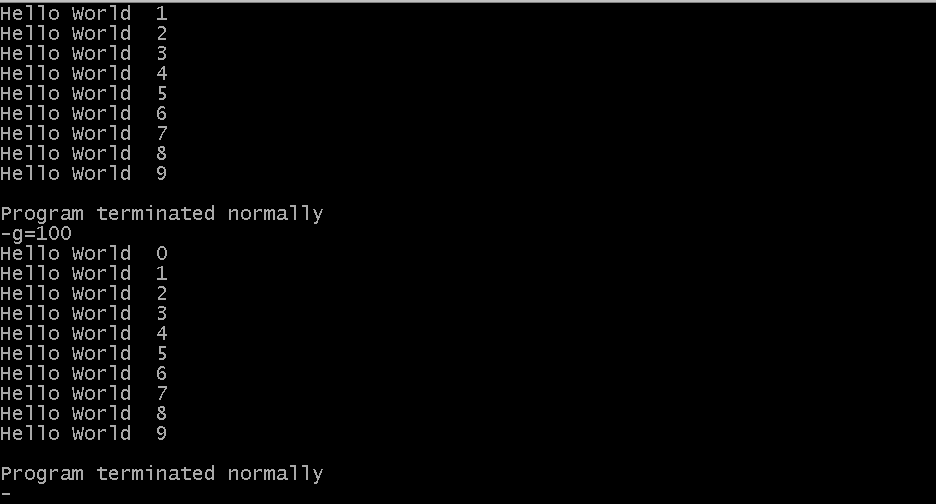
One of the issue with the coding was that it couldn’t be able to run twice, doing so will mess up the number. Reason being is that the number might be saved in the register, so when you output it the second time it would just increase from there.



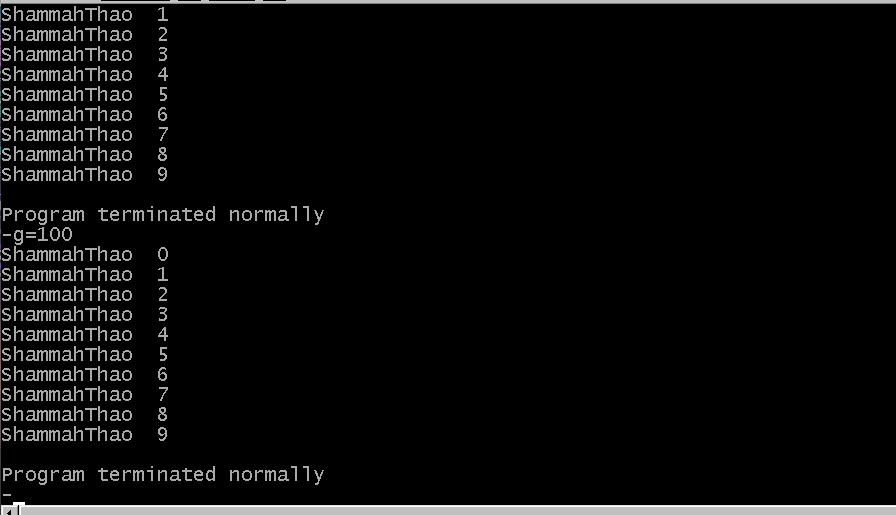
To fix the issue we had to fix the coding so it would reset back to 0 after the first loop.



Doing so we were able to get the program to run without any issues

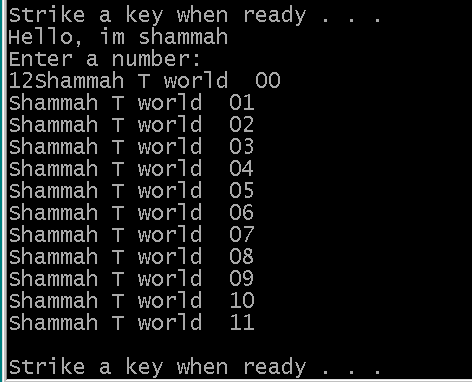


After fixing the issue we changed the quote to our name and see how it run with different character length

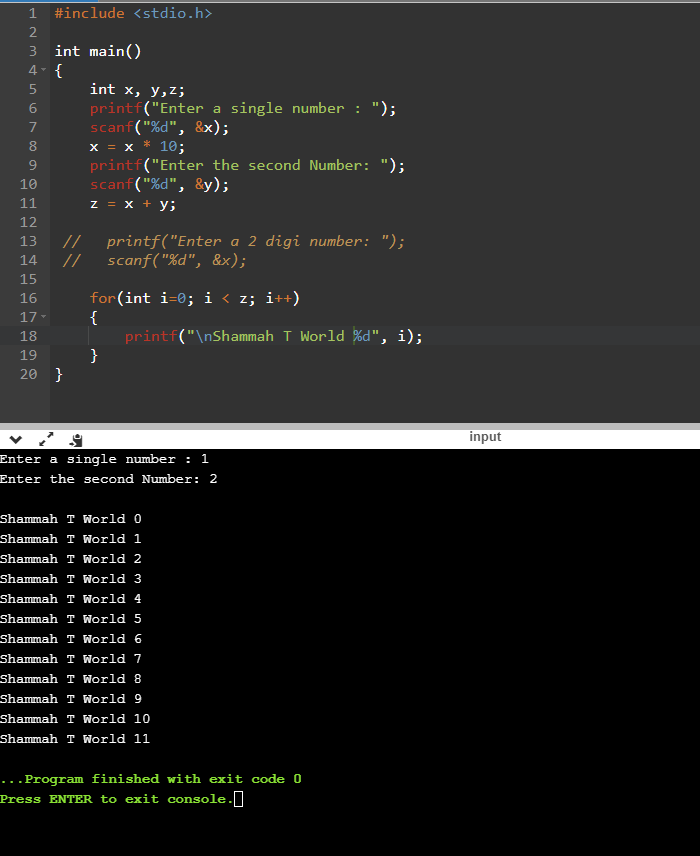


For the final part we had to make it so it takes in user input along with a title. We were able to make it so it output the correct number without much issue expect the issue with the input number and quote being on the same line.

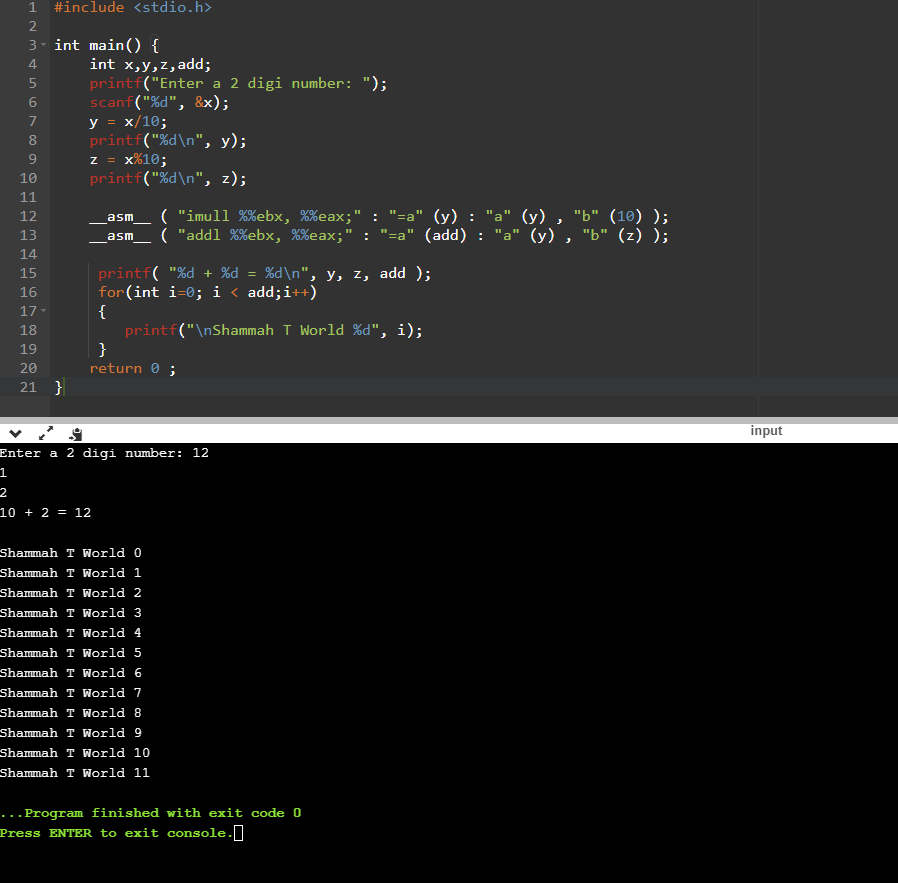




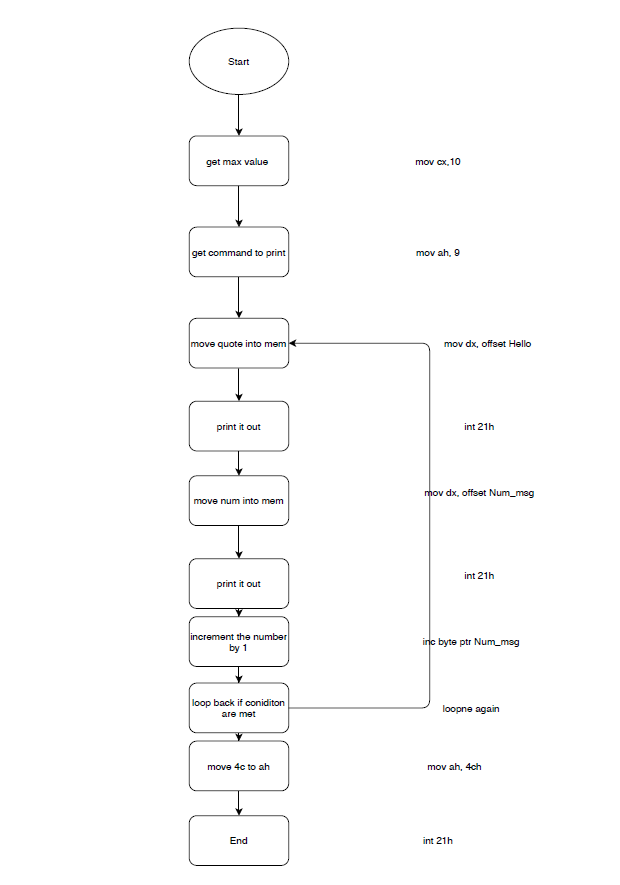
At this part, we made a pure C coding of what was happening in the pwd code. I make it so mine accept 2 single integer and add them together to get a 2 digit num and put it inside a for loop to run.



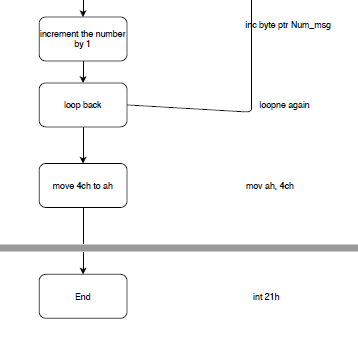
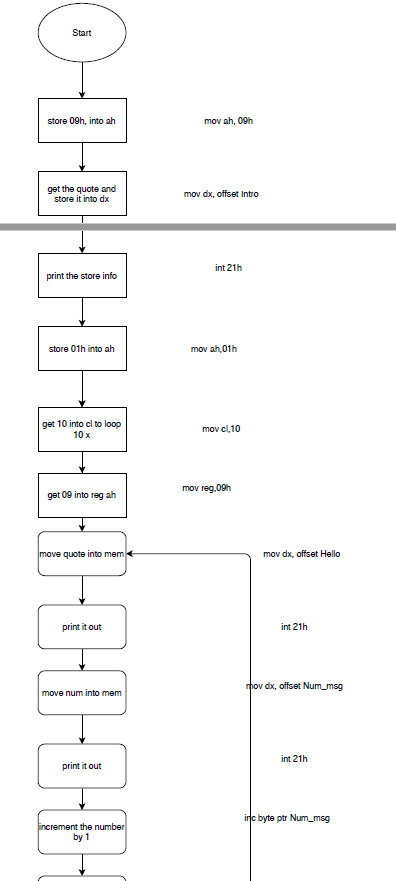
For this part of the coding, we had to put inline assembly inside C. So to do this we have to use \_\_asm\_\_ along with some assembly code. I made it so mine teared apart a 2 digit number then put in the asm which would multiply 10 to make it a 10 digit number, then add the second digit and then it was put inside a loop to print out the same output as the pwd.



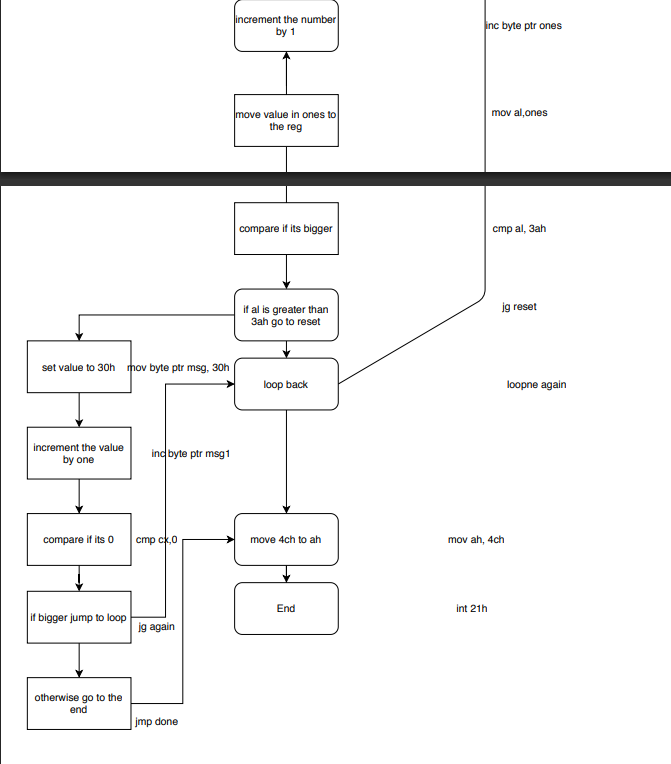
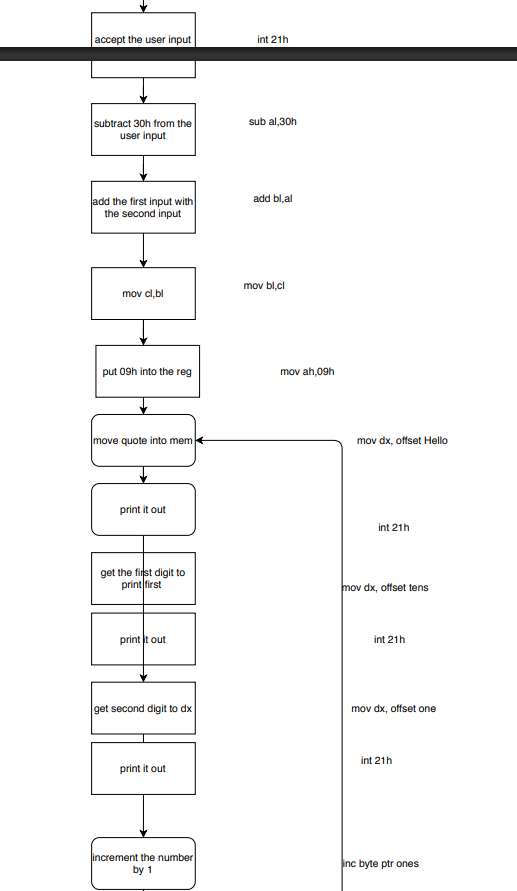
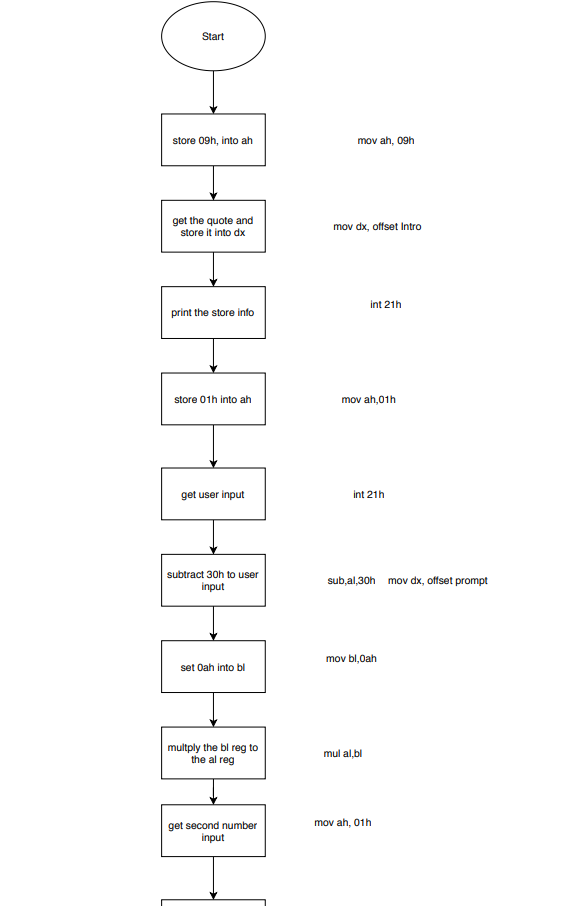
For the Flowchart of this lab, we have to include 3 flow chart. One with just the original code method, second is including the title in it, and lastly including the user input and incrementing the number.



The flowchart above is the original method of how this pwd program would run



The Second flow chart, above this is the method that included the incrementing and the title.



Lastly for the flow chart was the one that takes the user input, along with incrementing and title.

**Conclusion:**

In conclusion this lab was both time consuming and kind of tedious, abut at the same time it does help you learn about the assembly language debugger. It helps me understand how machine language come into play with hex, binary and from assembly instruction. Working with how the problem runs and how they should be executed. Using the debugger was great way to practice and understand what was going on during the execution of the code.