Nathan Tomlin
Layth Maloul
CISP 2410, Project 2
ntomlin@volstate.edu

SECTION 1 - OBJECTIVES

The objective of this project is to design, create, test, and debug an assembly language guessing game using MARIE SIM. In this guessing game, the magic number is hardcoded into the project, and the user must input a number into the console. When the user inputs a number into the console, the program will compare it with the magic number, and display a message as a direct result of this comparison. The messages are as follows; for a user input less than the magic number, the program will read "Too Low," for a number greater than the magic number, the program will output "Too High," and for a number entered that is equal to the magic number, the program will output "You Win." Our approach to solving this problem involved using concepts we have learned in computer organization and architecture, as well as logical thinking in order to break down the large pieces of this project into small, workable subsections. To start, we took all of the character arrays and converted them into their ASCII equivalent code using the ASCII table. We then specified a starting memory address for each of these "Strings" and hardcoded them into the program. Next, we designed the user input section, and began to branch out from there. Functions were made in order to check if the number was too high, too low, or correct. Each of these functions contained a jump statement to their respective print functions, in which the program would output the correct message for each function. Finally, everything was tied together with jump statements, which allowed us to be able to loop back to functions that needed to be repeated multiple times.

SECTION 2 – DESIGN DESCRIPTION

Regarding the high-level operation of this program, solutions were acquired based on two underlying objectives. The first was that there were to be 3 comparisons, with each comparison being linked to one another. We began by designing a comparison called check High to check if the number the user entered was greater than the magic number. The basis of this was that the magic number would be subtracted from the user input Using skip condition 800, which will skip the next instruction if the result of the arithmetic is greater than 0. If the skip condition was true, then the number was too high, which forced the program to jump to a loop in which the character array "Too High" was printed out to the user (output high & print high), and the user was sent back to the user input function a new number. If the arithmetic was not positive and the input was not greater than the magic number, then the program

jumped to another comparison to check if the number was less than the magic number (check low). In this comparison, a skip condition 000 was used, in which the magic number was once again subtracted from the user input. If the result of this arithmetic is less than 0, then the skip condition was true, and the user was sent to another loop to output the message “Too Low” before being sent back to enter another input. If the resulting arithmetic was not less than 0, then the program jumped to a final function called you win. The underlying idea of the you win function is that since we have already checked if the user input is greater or less than the magic number, then the only remaining condition must be that the user input equals that of the magic number. In this function, a loop was used to print the character array containing “You Win” to the user, and the program was halted.

SECTION 3 – TESTING AND RESULTS

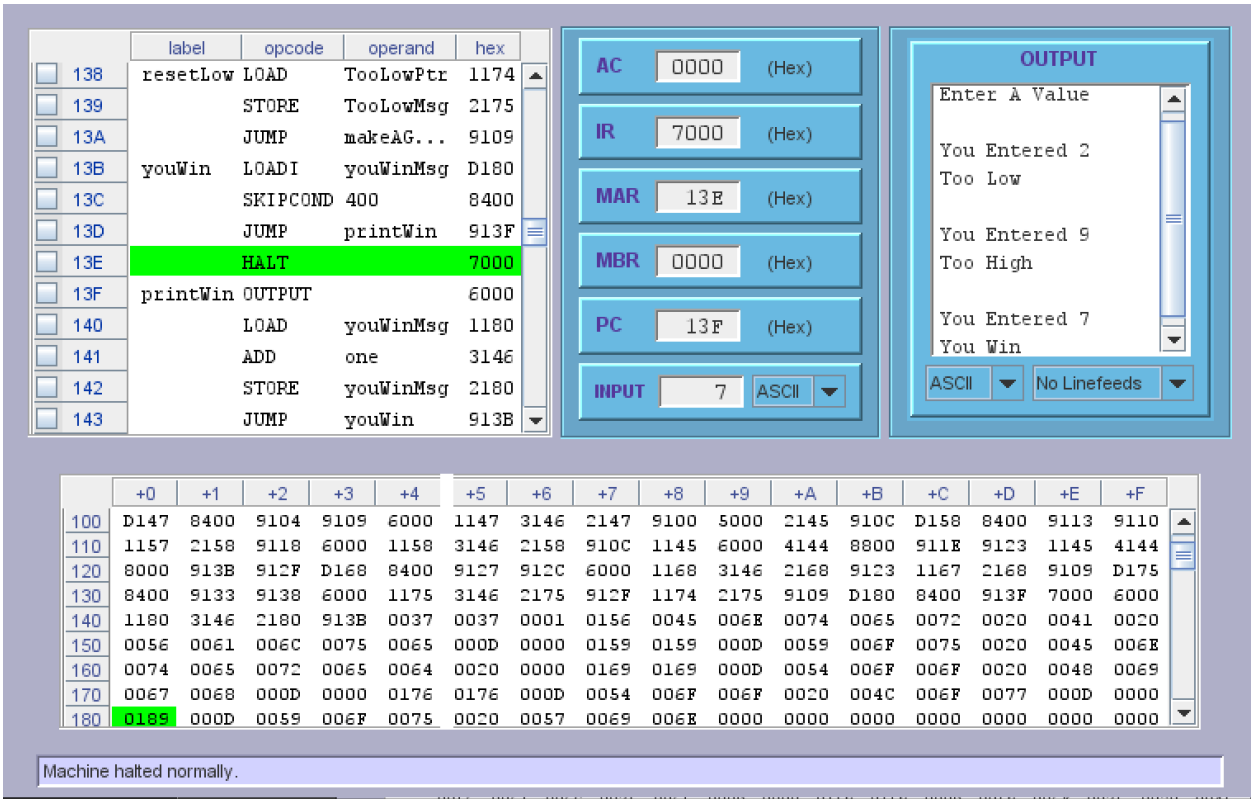


FIG 1.1

The output above is the result of multiple tests in order to verify that every aspect and design element of the program is working as intended. The first thing that is presented to the user is a loop that prints ever character in an array called enter value. This is presented to the user as the first thing they see upon running the program. Its design is as follows.

```
valueMsg,       LOADI           enterValue              /Loads the value associated with the starting memory address of enterValue
                SKIPCOND        400                     /Check if were at end of the array since the last value in array is 000
                JUMP            printMsg1               /Jump to the printMsg1 where we will output the message one array index at a time if the array is not at the end
                JUMP            makeAGuess
printMsg1,      OUTPUT                                  /Outputs the  value associated with the memory address that was loaded into accumulator
                LOAD            enterValue
                ADD             one                     /Loads the memory address itself, incremints it by 1 which moves it to the next address in the array
                STORE           enterValue              /Store this new address at enterValue
                JUMP            valueMsg                /Jump back to top in order to repeat until all values of loop have been printed
```

FIG1.2

Enter value is an array whose contents were hardcoded into memory using MARIES hex keyword, followed by the ASCII codes for each character, including space and new line. In this group of subroutines, the starting value of the memory address of the array is loaded into the accumulator. The value is then compared with a skip condition 400, which will jump out of the loop to the make a guess subroutine if the value associated with memory equals 0, which works because the last value in this array is set to 0. While the value associated with memory is not 0, the function(s) will output that value, increment the memory address of the starting position by one, and jump back to the top until the entire array is printed. Once the array is printed, the program will jump to the make a guess subroutine in order to let the user input a value.

```
makeAGuess,     INPUT
                STORE           userInput
                JUMP            youEntered

youEntered,     LOADI           youEnterMsg             /Loads the value associated with the starting memory address of youEnterMsg to display "You Entered"
                SKIPCOND        400                     /Check if were at end of the array since the last value in array is 000
                JUMP            printYouMsg             /Jump to the printYouMsg where we will output the message one array index at a time if the array is not at the end
                JUMP            resetEnter              /Once we reach the end of the array, we will jump tp resetEnter, in order to reset the array back to its starting address in memory
resetEnter,     LOAD            youEnterPtr             /Loading memory address of pointer pointing to the starting address of the youEnterMsg array
                STORE           youEnterMsg             /Setting the memory address of the array youEnterMsg equal to its starting address, so when this is called again we can print the same message
                JUMP            checkHigh               /We will now jump to checkHigh  to begin checking the value to see if it is too high
```

FIG 1.3

Once the user inputs a value into the input window in MARIE, that value is stored inside of a variable in memory named user input. The make a guess function then jumps to the you entered function, which will use the same design as the one shown before to print out the array of characters “You entered” to the screen. Once the array has been printed, the function jumps to a subroutine called check high in order to start the process of comparing the value of the users input with that of the magic number.

```
checkHigh,      LOAD            userInput               /Loading to output guess then check if value is positive. If it is, jump to outputHigh, if not jump to checkLow
                OUTPUT                                  /Outputs what the user entered as their guess
                SUBT            magicNumber
                SKIPCOND        800
                JUMP            checkLow                /Jumping to check and see if the number is negative which means it is too low
                JUMP            outputHigh              /Jump to output the message "Too High"
```

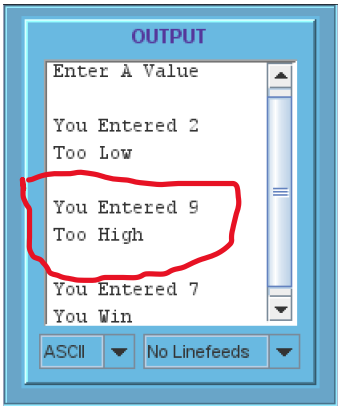
FIG 1.4

This begins the three comparisons that use conditional logic in order to see if the user entered the correct number. Check high subtracts the magic number from the number entered by the user. Using a skip condition 800 allows us to verify the results of the arithmetic. Logically thinking, if the result of this arithmetic is positive, then the user input was greater than the magic number. The function will then jump to a subroutine which contains a loop in order to print the message “Too High” named output high (Pictured Below).

```
outputHigh,     LOADI       TooHighMsg          /Loads the value associated with the starting memory address of TooHighMsg to display "Too High"
                SKIPCOND    400                 /Check if were at end of array since last value in array is 000
                JUMP        printHigh           /Jump to printHigh where we will output the message one index at a time if the array is not at the end
                JUMP        resetHigh           /Since we may need to print this more than once, jump to resetHigh to set array back to its starting point in memory
printHigh,      OUTPUT
                LOAD        TooHighMsg          /Loads the memory address itself, incremints it by 1 which moves it to the next address in the array
                ADD         one
                STORE       TooHighMsg          /Store this new address at TooHighMsg
                JUMP        outputHigh          /Jump back to top in order to repeat until all values of loop have been printed
resetHigh,      LOAD        TooHighPtr          /Loads a pointer that points to the starting address of the array TooHighMsg
                STORE       TooHighMsg          /Reset array by setting TooHighMsg equal to the memory address of TooHighPtr, which is the starting address of the arrray.
                JUMP        makeAGuess
```

FIG 1.5

This group of functions are designed to print the message to the user, and then reset the character array to its starting point in memory so that the message can be printed again in the future if the user enters another number that is too high.



In this case, the magic number was 7. You can see here that the entered a value of 9. Subtracting the magic number (7) from the input of the user (9) would give us a positive number greater than 0. As a result, the function above is called, and too high is printed to the screen.

FIG 1.6

It is important to remember that this function is only accessed if the result of the arithmetic in the check high function is positive. If the result of the arithmetic is not positive, then further testing must be done in order to solidify the outcome of the users guess. This is done by jumping to another function called check low, which will see if the users guess is less than that of the magic number.

```
checkLow,       LOAD        userInput           /Checking here to see if the userInput is too low by subtracting the magic number from it.
                SUBT        magicNumber
                SKIPCOND    000                 /If the result is negative, the userInput is too low, if not the user wins the game
                JUMP        youWin              /Since we already checked if the userInput was too hogh or too low, then the only remaining condition is that the user guessed the right number.
                JUMP        outputLow           /If the number is too low, jump to outputLow to display the message "Too Low"
```

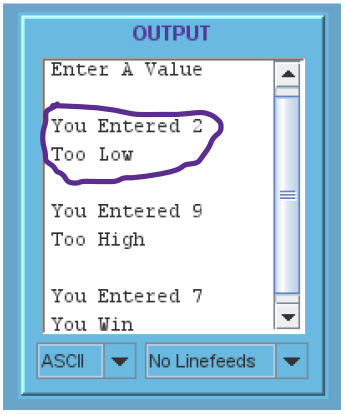
FIG 1.7

Check low subtracts the magic number from the number entered by the user. Using a skip condition 000 allows us to verify the results of the arithmetic. Logically thinking, if the result of this arithmetic is negative, then the user input was less than the magic number. The function will then jump to a subroutine which contains a loop in order to print the message “Too Low” named output low (Pictured Below).

```
outputLow,      LOADI           TooLowMsg           /Loads the value associated with the starting memory address of TooLowMsg to display "Too Low"
                SKIPCOND        400                 /Check if were at end of array since last value in array is 000
                JUMP            printLow            /Jump to printLow where we will output the message one index at a time if the array is not at the end
                JUMP            resetLow            /Since we may need to print this more than once, jump to resetLow to set array back to its starting point in memory
printLow,       OUTPUT
                LOAD            TooLowMsg           /Loads the memory address itself, incremints it by 1 which moves it to the next address in the array
                ADD             one
                STORE           TooLowMsg           /Store this new address at TooLowMsg
                JUMP            outputLow           /Jump back to top in order to repeat until all values of loop have been printed
resetLow,       LOAD            TooLowPtr           /Loads a pointer that points to the starting address of the array TooLowMsg
                STORE           TooLowMsg           /Reset array by setting TooLowMsg equal to the memory address of TooLowPtr, which is the starting address of the array.
                JUMP            makeAGuess
```

FIG 1.8

These functions will work together to print the message “Too Low” to the user, by setting up a loop that iterates over the character array too low msg. Much like the output high function before, this group of functions will also reset the character array to its starting point in memory so that the message can be printed again in the future if the user enters another number that is too low.



In this case, the magic number was 7. You can see here that the entered a value of 2. Subtracting the magic number (7) from the input of the user (2) would give us a negative number less than 0. As a result, the function above is called, and too low is printed to the screen.

FIG 1.9

Like before, the functions to print too low are only accessed if the result of the arithmetic in the check low function is less than 0. If the value of that arithmetic is not less than 0, then we jump to our last function called you win.

```
youWin,         LOADI           youWinMsg           /Load the value associated with the memory address of the youWinMsg array to display "You Win"
                SKIPCOND        400                 /Check if were at the end of array since the last value in array is 000
                JUMP            printWin            /Jump to printWin to begin outputting the message one index at a time, until we hit the end of the array
                HALT                                /Once array is fully printed, skipcond will activate and we will halt the program, as we have done everything
printWin,       OUTPUT
                LOAD            youWinMsg           /Inreminting memory address of youWinMsg, in order to get the next value from memory
                ADD             one
                STORE           youWinMsg           /Storing this incremented value so we can reference it the next time we loop
                JUMP            youWin              /Jumping back to youWin, so we can repeat until finished
```

FIG 2.0

The reason that this is the last function of the program is simple. In order to get to get to you win, the user input must first pass through check high and check low. If the value is not greater than the magic number, and then value is not less than the magic number, than the only possible conclusion is that the value is that of the magic number. In this function, the message “You Win” is printed to the screen as an array of characters. Notice we do not have to reset this array back to its starting position in memory, and that is because it will be the last thing that is called before the program halts meaning that it will only be printed once.

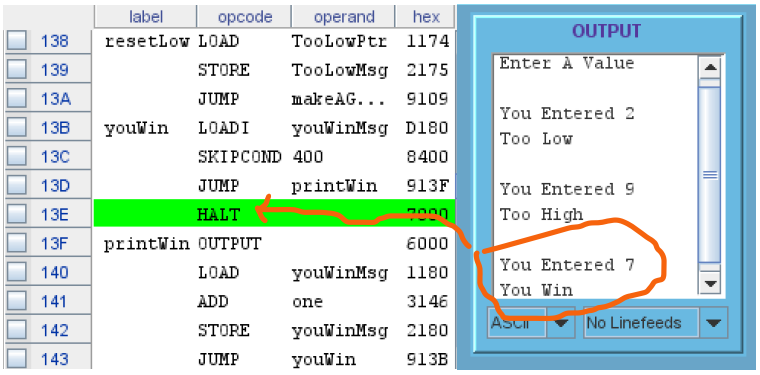


FIG 2.1

More testing can be seen below, in which other test cases are executed to ensure the program is working as intended.

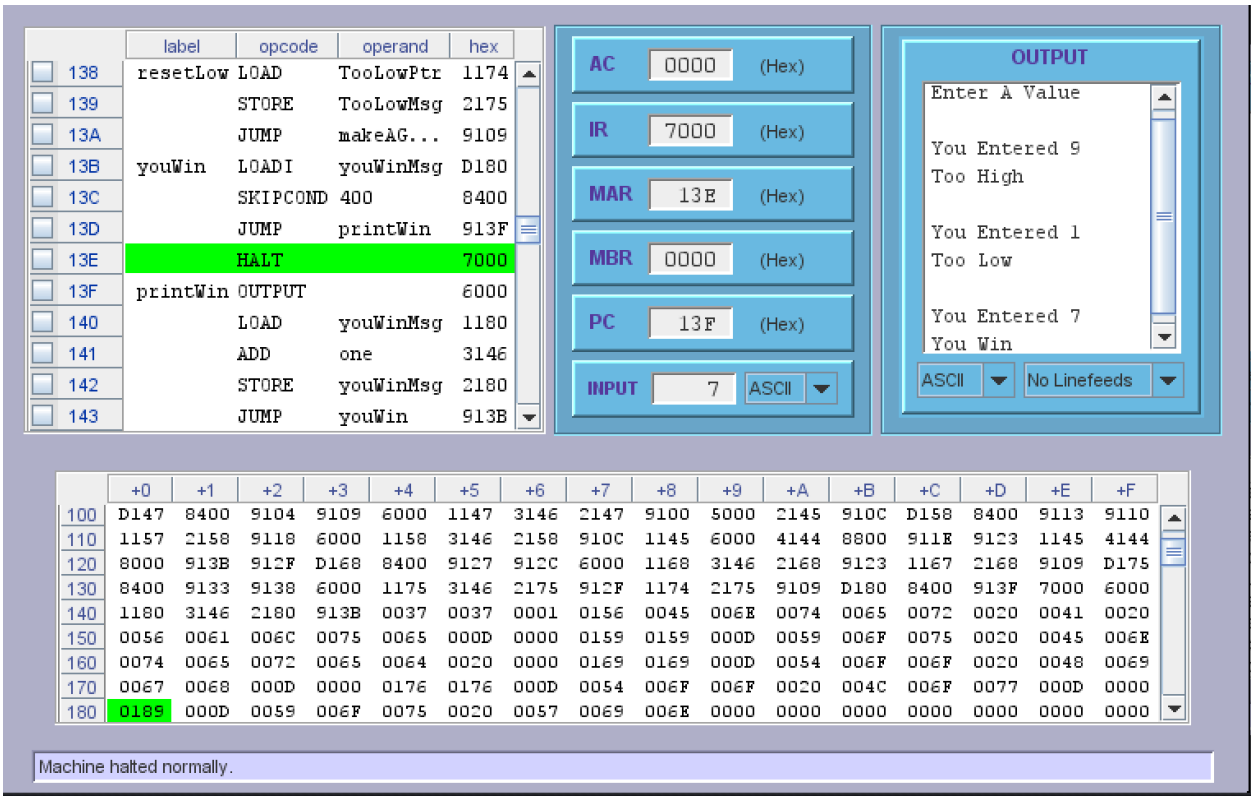


FIG 2.2

In this test case above, the order of the too high and too low tests were swapped. Testing a number too high first and testing a number too low after yielded no errors or cause for concern.

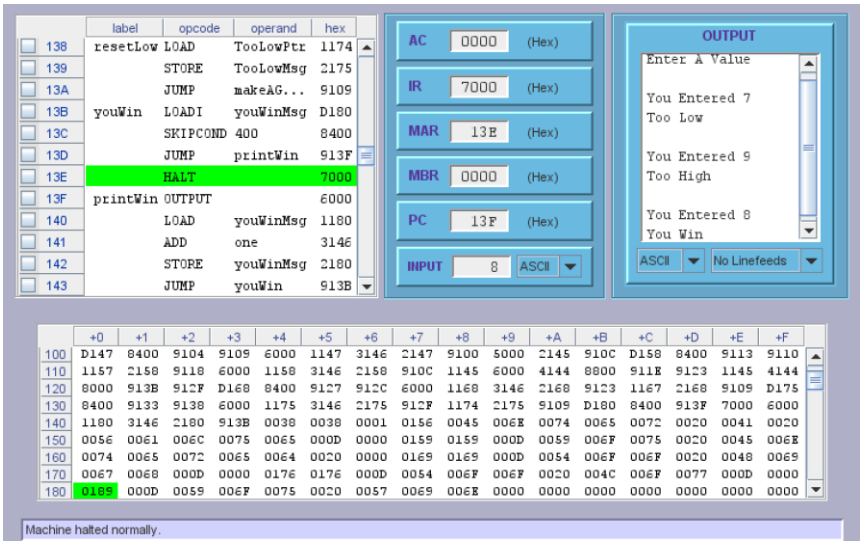


FIG 2.3

It should be noted that changing the magic number does not change ANY functionality of the program. When considering all of the previous testing in combination with the logical steps to arrive at this final product, it can be safely assumed that the program works exactly as intended, with no logical errors whatsoever.

SECTION 4 - CONCLUSIONS

The outcome of the design created specifically for this project is a fully functional, logically correct program that successfully allows the user to guess against the machine. The program works in all aspects, with no underlying fundamental flaws in the logic and or design. One of the most important things we have learned from this project is just how important it is to take an overarching goal and break it down into smaller, less stressful subsections that flow cohesively and build off one another. It is also important to realize the significance of MARIE SIM, more specifically assembly language, and how communicating directly with the computer has advantages as well as disadvantages. Using ASCII in combination with MARIE SIM provided some initial difficulty, but after further inspection the problem was quite trivial. Overall, this project was incredibly helpful with understanding logic and assembly language, and how the two go hand in hand together.