*Name: Trung Kien Nguyen*

*Student ID: 104053642*

*From Lab 18 (10:30 a.m to 12:30 a.m on Friday)*



COS10004

COMPUTER SYSTEMS

Report

Assigntment 2



**Table of contents**

*Table of contents…………………………………………………………………………………………………….…2*

*Introduction………………………………………………………………………………………………………………3*

*Description………………………………………………………………………………………………………………..4*

*Stage 1……………………………………………………………………………………………………………..4*

*Stage 2……………………………………………………………………………………………………………..4*

*Stage 3……………………………………………………………………………………………………………..5*

*Stage 4……………………………………………………………………………………………………………..6*

*Stage 5……………………………………………………………………………………………………………..7*

*5a…………………………………………………………………………………………………………….7*

*5b…………………………………………………………………………………………………………….9*

*Stage 6……………………………………………………………………………………………………………..9*

*Conclusion……………………….……………………………………………………………………………………...10*

*References…………………………….…………………………………………………………………………………10*

**Introduction**

*Hi, I’m Trung Kien Nguyen, student ID 104053642, now I’m studying COS10004 Computer Systems, at lab 18 from 10:30 am to 12:30 am on Friday weekly.*

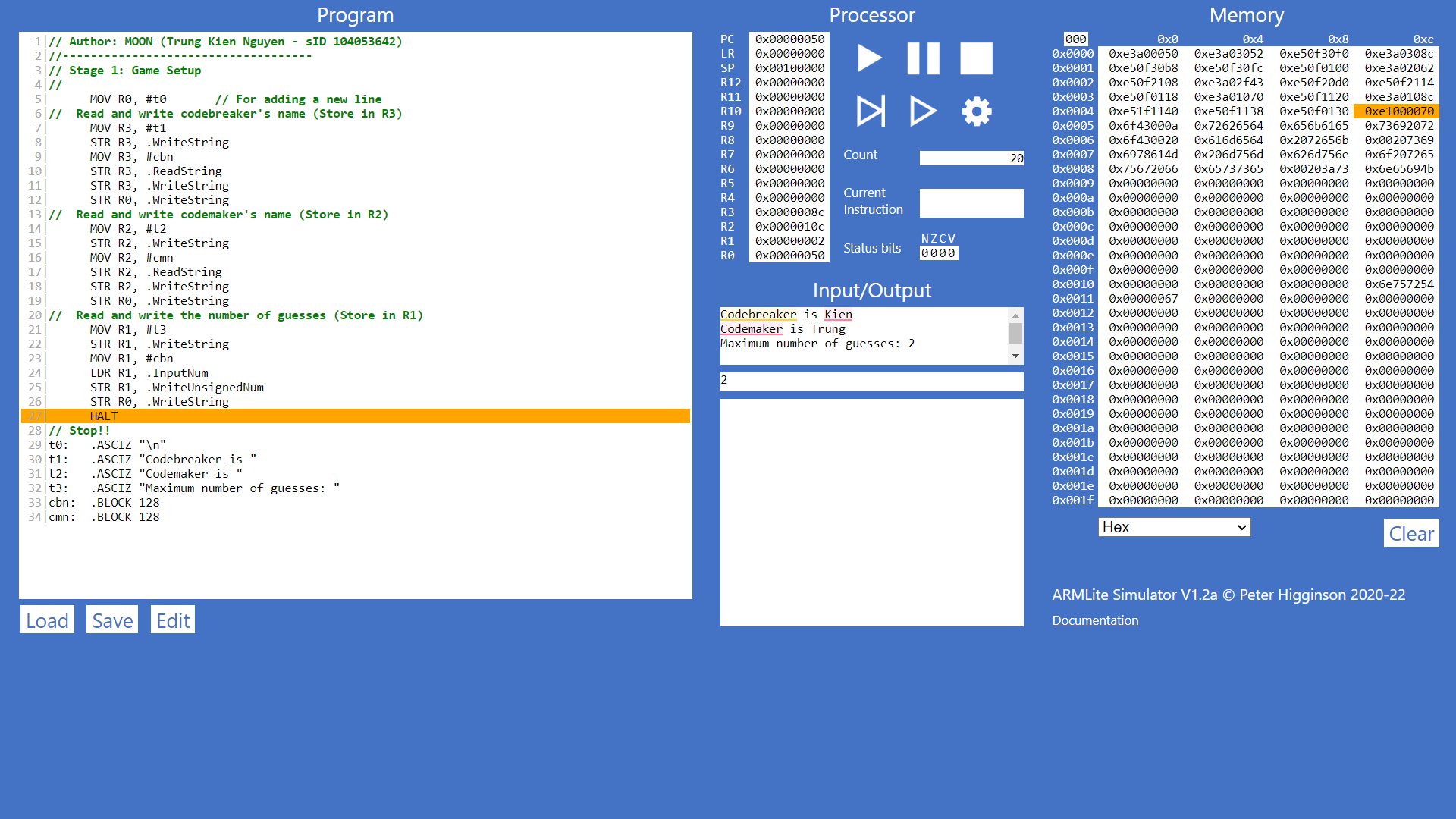
*This is my report for Assignment 2 – ARM Assembly Programming - Mastermind, which will provide a clear description of my work in the recent weeks.*



**Description**

*My work consists of 7 files, corresponding to each stage of the assignment. The file of the later stage includes all parts of the previous ones (with some slight suitable changes), and the file stage6.asm contains the complete program.*

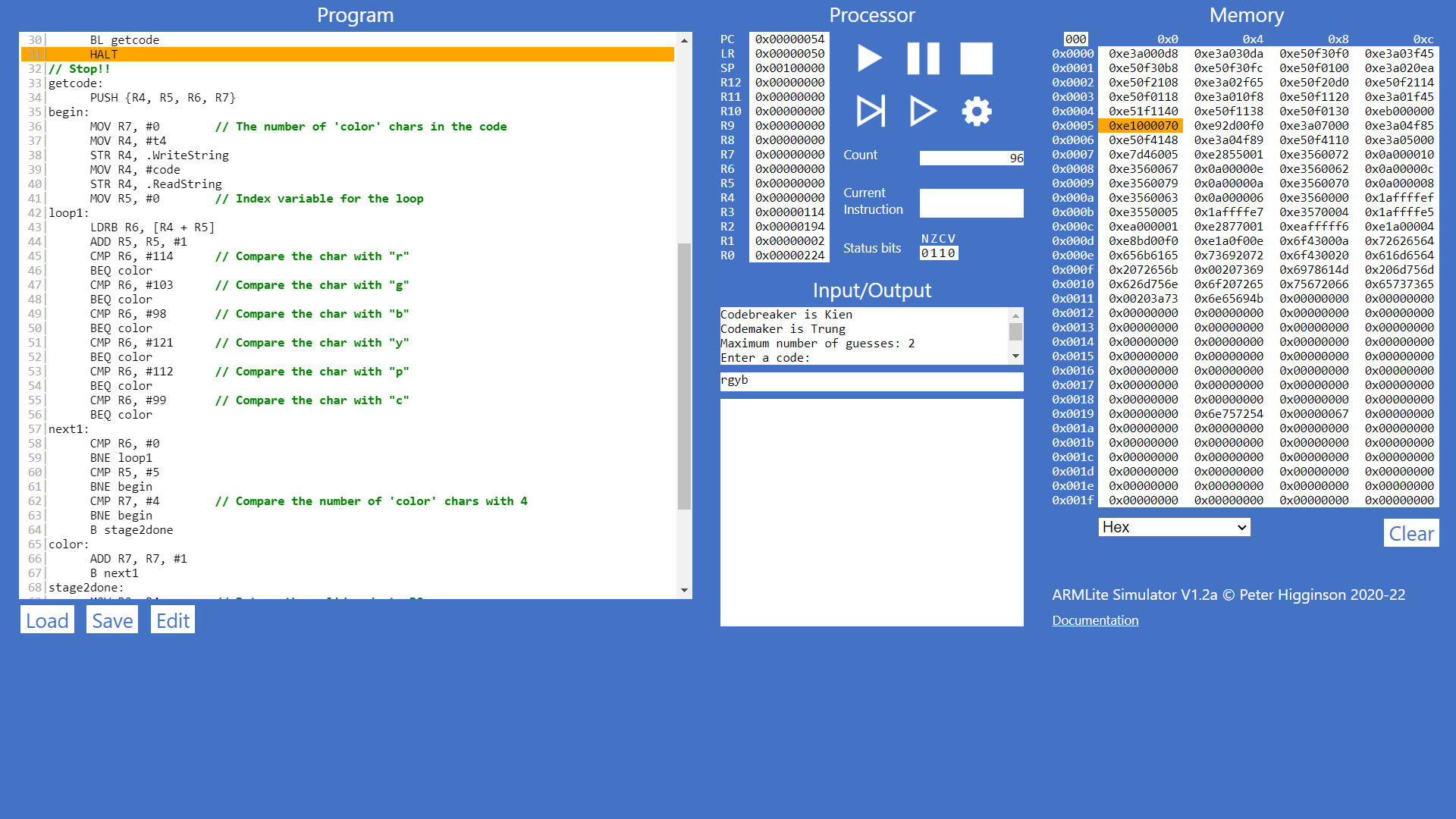
Stage 1: Game setup



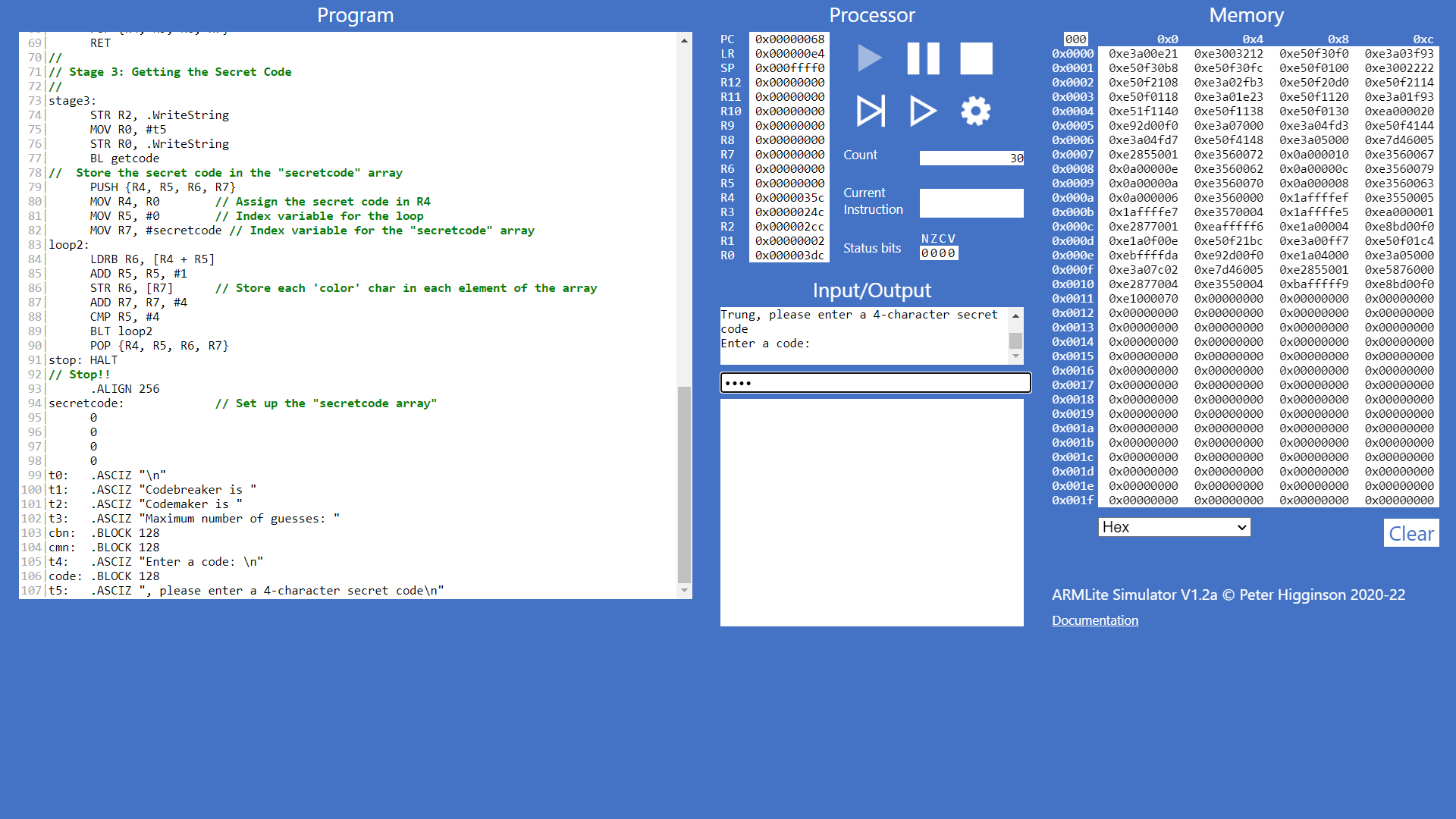
In this stage, I simply assign the strings of instruction to R3, R2 and R1, and then allow the player to input and assign the value of codebreaker’s name, codemaker’s name and maximum number of guesses repectively.

Stage 2: Code entry function (“getcode”)

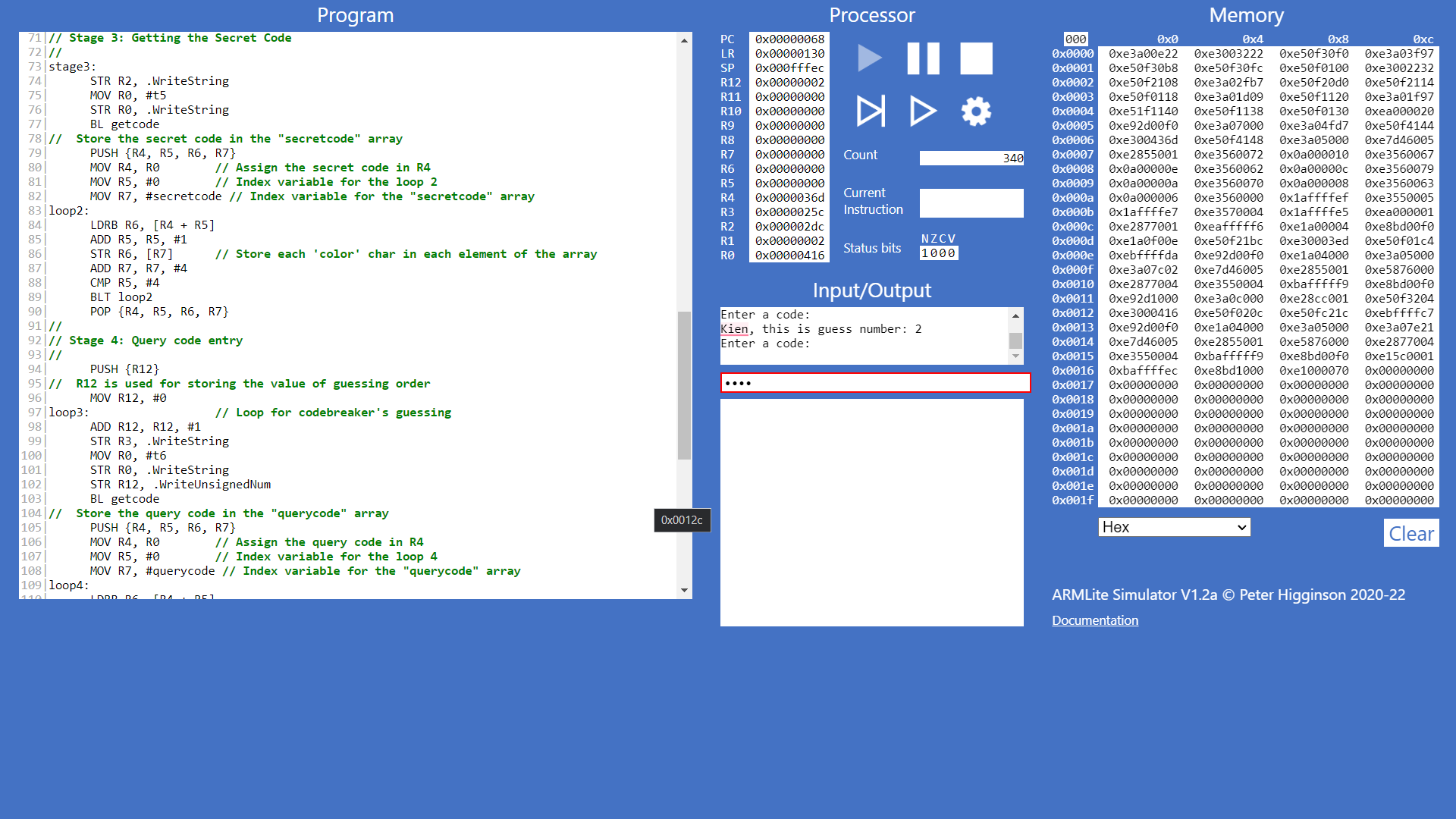
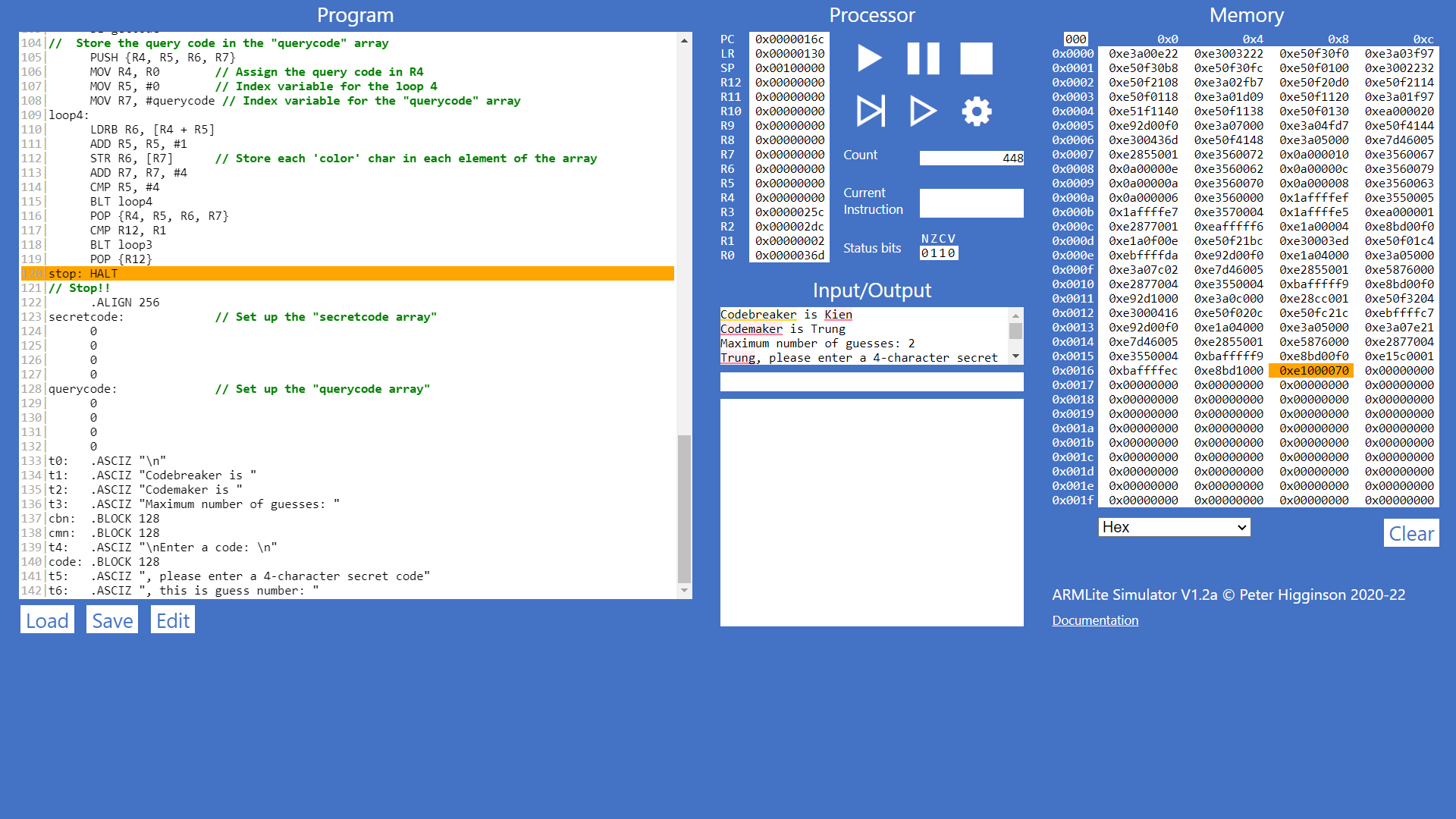
The function names “getcode”, initially asks the player to enter a code, then go to the loop (“loop1”). Each character of the entered code will be assigned to R6, and R5 will increase by 1, then will use the branch instruction to check that the character belongs to one of the 'r', 'g', 'y', 'b', 'c' and 'p' characters or not (refer to ASCII table), if yes, it will increase R7 by 1. This is repeated until the last character. At the end of the loop, R5 will be the number of characters of the code, and R7 will be the number of characters that are 'color', then the program only needs to check if R5 and R7 are exactly equal to 4 or not, if so, it will exit the method with the “RET” instruction, otherwise return to the beginning. The valid code will be returned to R0, and then the program uses the “POP” instruction to restore all of the R4, R5, R6, and R7 according to the ABI conventions.



Stage 3: Getting the Secret code

 **From this stage, I will change slightly the “getcode” function, from “.ReadString” to “.ReadSecret” command.** After entering the inputs (in stage 1), the program will go directly to stage 3 (“stage3”). First, it outputs the instruction that requires the codemaker to enter a secret code (R2 + “t5”) and calls for the “getcode” function. After this function returns the entered valid code to R0, it begins to store each character of the code in the “secretcode” array. To do this, I use R7 as the address to the array. In the “loop2”, each character of the code will be assigned to R6, then stored in the memory (address of R7) by the instruction of “STR R6, [R7]”. After the loop ends (4 characters have been stored), I use the “POP” instruction to restore all of the used registers (including R4, R5, R6, and R7), to follow the ABI conventions.

Stage 4: Query code entry

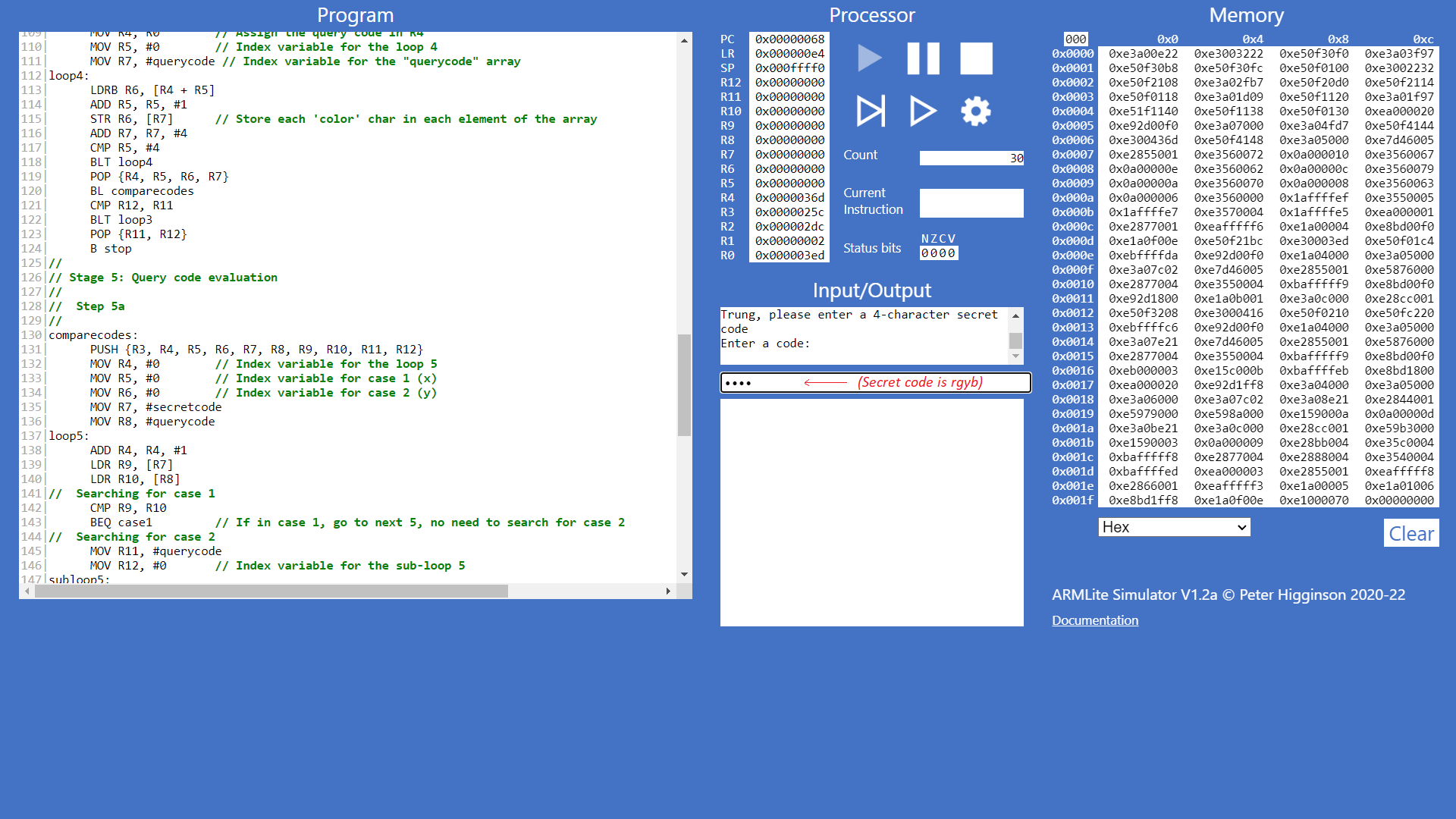
After stage 3 has been done, the program executes to stage 4. This stage is simply the number of stage 3, but the instructions require codebreaker to enter. This uses a loop (“loop3”), from 1 to the number of guesses (R12 is the index value). Each time, it outputs the instruction that requires the codebreaker to enter a query code (R2 + “t5”) and calls for the “getcode” function. After this function returns the entered valid code to R0, it begins to store each character of the code in the “querycode” array (with the same procedure of storing in stage 3, but using the “loop4”). Similarly, after this stage ends, the “POP” instruction will be used to restore the used registers, except R0, R1, R2, and R3.

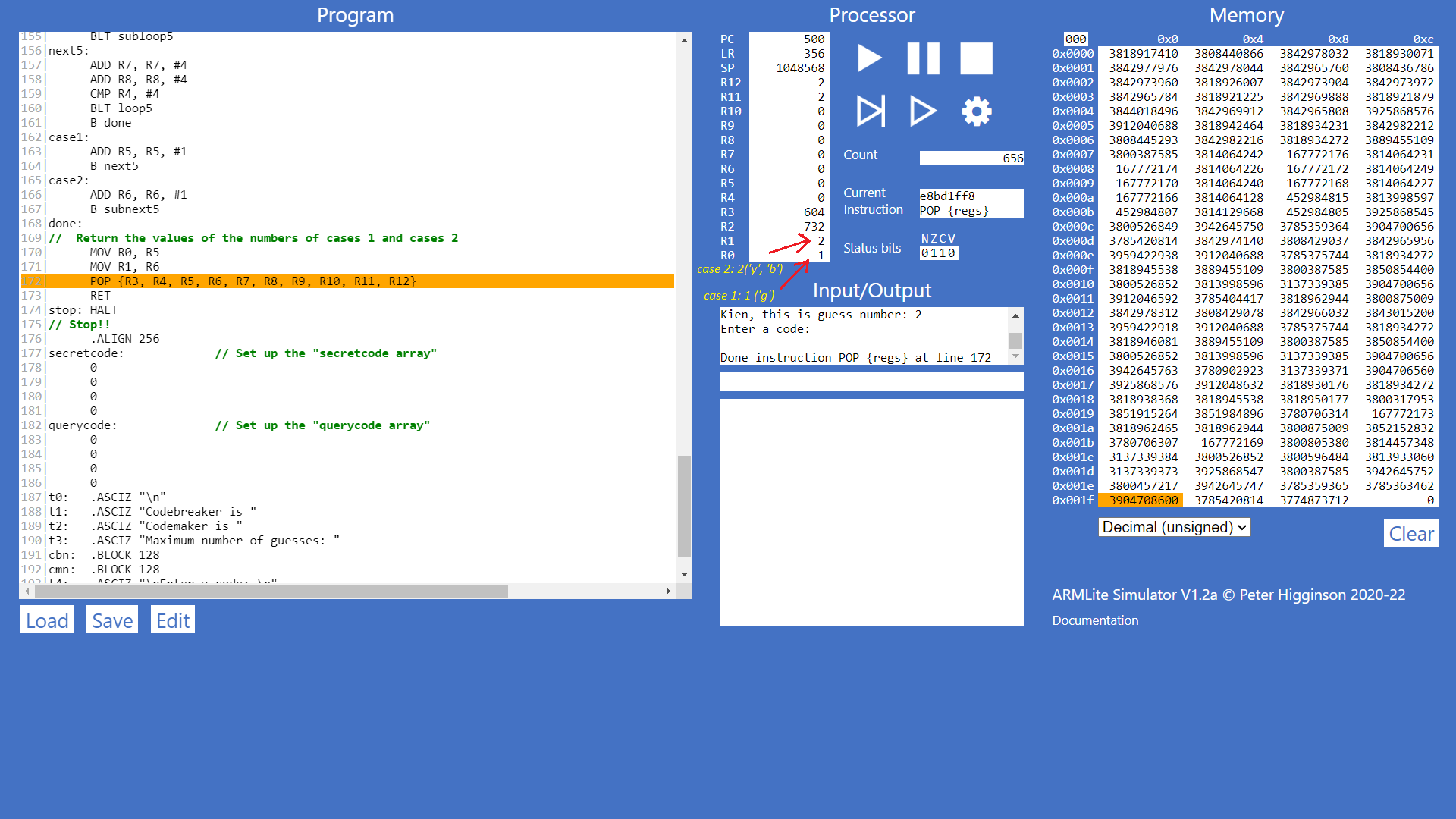
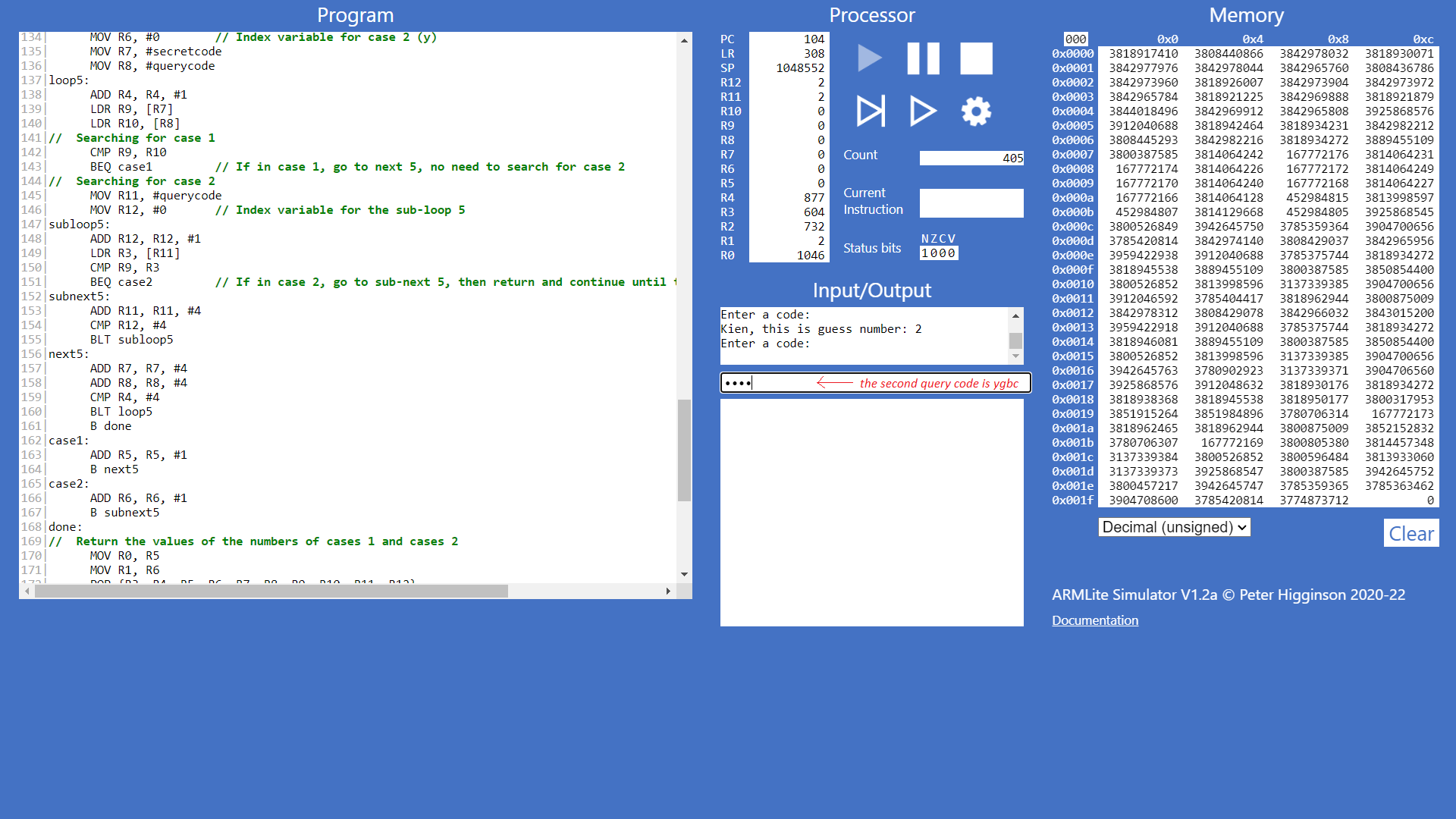
Stage 5: Query code evaluation

**I have made a slight change at this stage in stage 4’s code, in storing the maximum number of guesses. It is now R11 because R1 is used for storing the number of cases 2 followingly.**

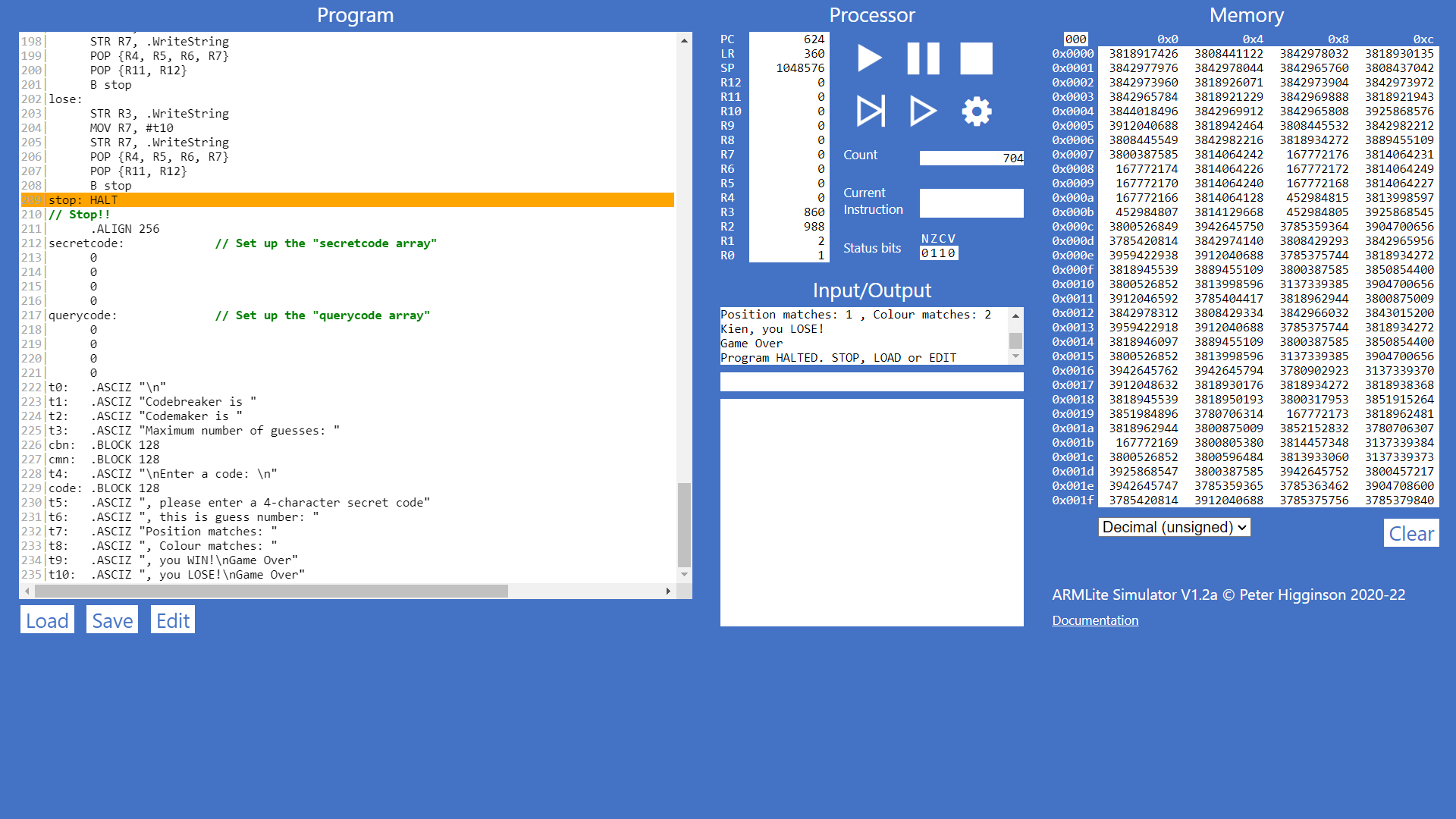
(5a) Compare query and secret code:

The “comparecodes” function has been made and added every time the valid query code is stored. In this function, I use a loop (“loop5”) (R4 is the index for this loop), R5 and R6 are the numbers of cases 1 and 2, while R7 and R8 are the addresses of each character in “secretcode” and “querycode” respectively in the memory. Each time of "loop5", while R4 is less than 4, each character of the two codes will be loaded to R9 and R10, and the program will compare them. If they are equal, R5 will be increased by 1, if not, the subloop (“subloop5”) begins (R12 is the index for this subloop). The subloop will load all the characters of the “querycode” again respectively to R3 (while R12 is less than 4), and compare them to the current character of “secretcode” (stored in R9). If they are equal to each other, as it is not case 1, it must be case 2, which means R6 will increase by 1. This is repeated until "loop5" is over. After that, the function returns R5 and R6 (number of cases 1 and 2) to R0 and R1 respectively, and all used registers, including R3, will be restored from the stack (by "POP" instruction) (ABI conventions).





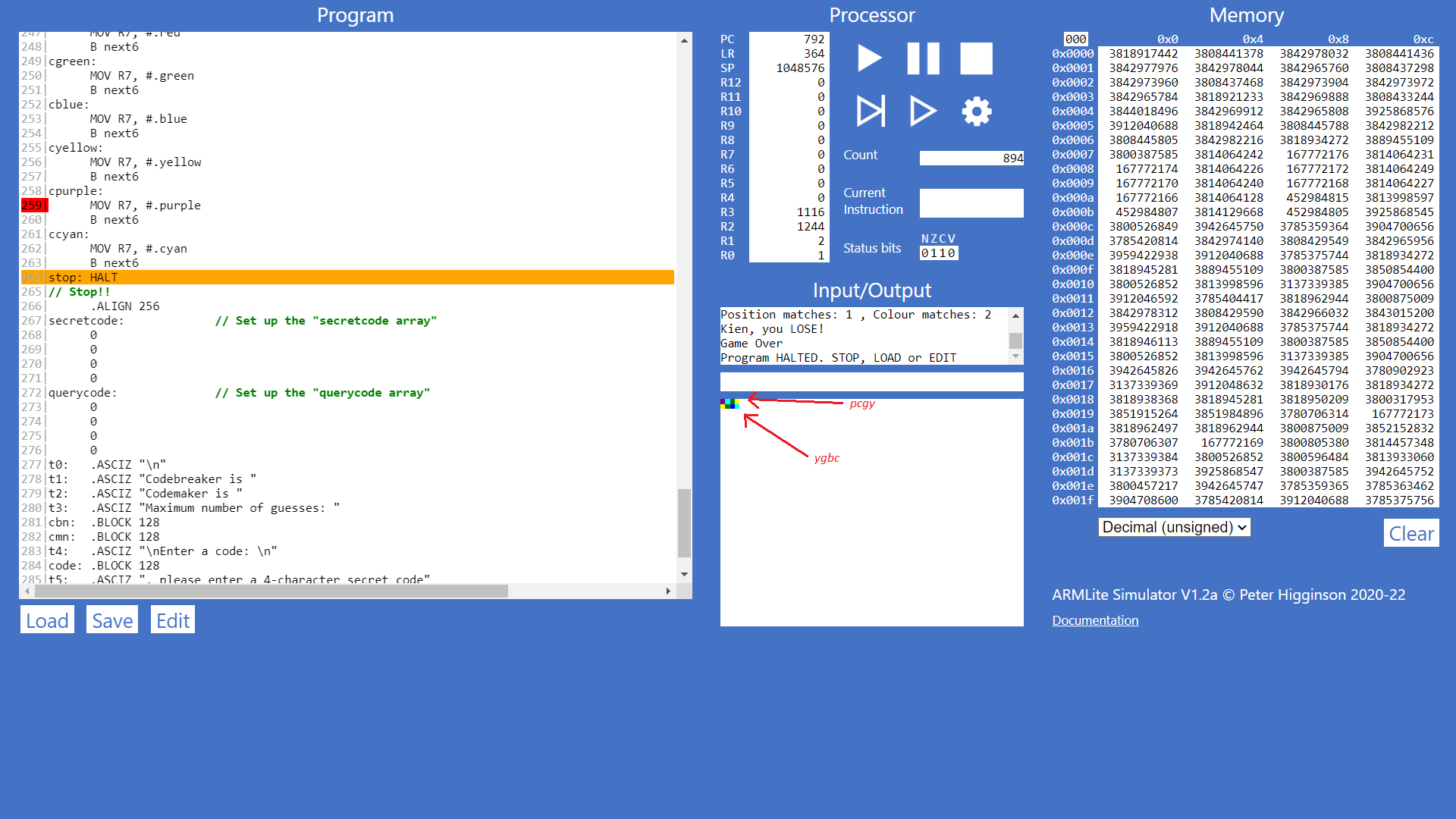
(5b) Code Evaluation Feedback and Game Termination

*(The inputs are still the same as above)*

At this step, the “feedback” function has been made and added every time the “comparecode” function is finished. Initially, the numbers of cases 1 and 2 (stored in R0 and R1) now are stored in R5 and R6, while R4 is used to store the number of times the codebreaker has guessed (stored in R12 before), then it simply prints the messages (“t7” + R5 + “t8” + R6, all are assigned to and output by R7 respectively). Next, the branch statements are conducted, if R5 (number of cases 1) is equal to 4, the codebreaker wins (the “win” is called), if not, the program continues the loop in stage 4 (“loop3”). If the number of times the codebreaker guessed (R4) is equal to the maximum number of guesses (currently R11) and the “win” has never been called, the “lose” will be called. For both these results, the program will print the codebreaker’s name (stored in R3), and “t9” (in “win”) or “t10” (in “lose”), then it will restore all the used registers from the stack by “POP” instructions (R4, R5, R6 and R7 in this function, and R11, R12 in stage 4).

Stage 6: Visualise query history (**This file (stage.asm) is my complete program**)

At this step, the “draw” function has been made and added, after storing the “querycode”, and before the “comparecode” function is called. In this function, R4 is the index of the loop (“loop6”) (each time increasing by 4) and pixel drawing position of each line as well, R5 is the address of each element of the “querycode” array, R6 is to store those elements (‘color’ characters), R7 is to store the respective colors, R8 is to use the graphic display in the Medium-resolution mode (“.PixelScreen”), and R9 is to decide which line will be drawn (Defined by: ( ((The index of current guess) – 1) \* 256) ). Next, the program goes into a loop (“loop6”), which deals with each character. Each time, it compares R6 with all the ASCII codes of 6 possible ‘color’ characters (114, 103, 98, 121, 112, and 99), and assigns the respective color to R7 (in “cred”, “cgreen”, …), before draws it to the graphic display in the right position, which is R10 = R9 + R4, by the instruction “STR R7, [R8 + R10]”. Finally, all the used registers for this function (from R4 to R10) will be restored to meet the requirement of the ABI conventions.

****

**Conclusion**

Above are all my solutions for all stages of the assignment 2, of which stage6.asm is my complete program for Mastemind boardgame.

Thank you for taking the time to read this report, if you have any questions about it, please contact me by sending an email to the address: [104053642@student.swin.edu.au](mailto:104053642@student.swin.edu.au)

**References**

*[1] Lectures and recordings from week 7 to week 10*

*[2] Lab Instructions from 7 to 10*