COS10031 Computer Technology

Assignment 3: ARMLite Mastermind Game

8:30am Tuesday, 10:30am Wednesday
with Dr. Sourabh Dani

Nicole Reichert (100589839)
Marcus Mifsud (105875038)
Vandy Aum (105715697)
Luke Byrnes (7194587)

Due: 18 May 2025

Diploma IT - Swinburne College

Table of contents

Mastermind Assembly Game

Program Overview

This program replicates gameplay of the Mastermind boardgame in Assembly using the ARM-Lite assembly utility.

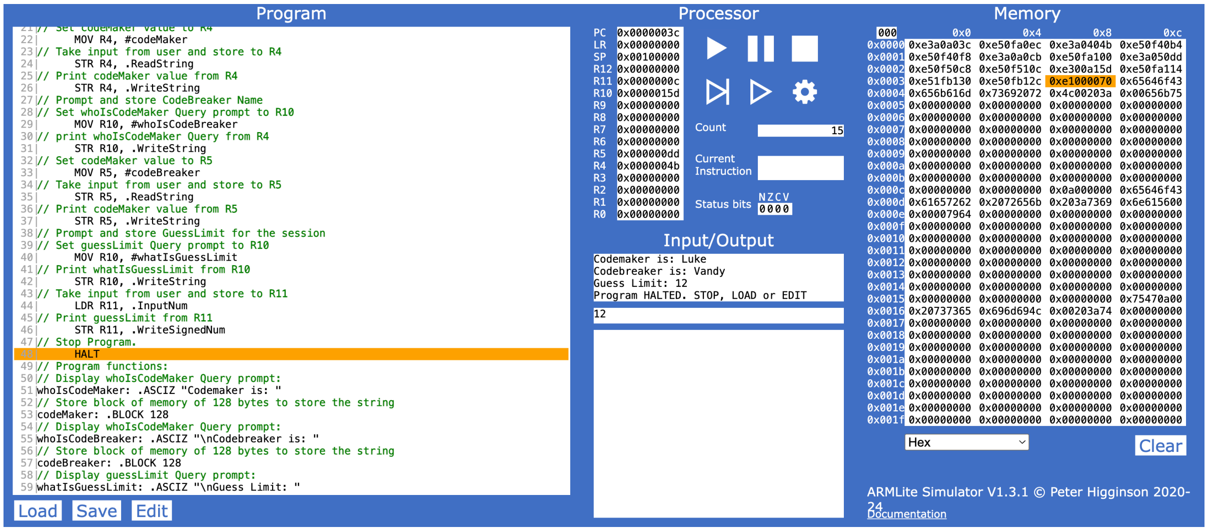
Key Functions

Stage 1 (`stage1.txt`)

Stage 1 makes use of the following functions:

Listing 1 Functions of 'stage1.txt'

```
// Program functions:
    // Display whoIsCodeMaker Query prompt:
    whoIsCodeMakerMsg: .ASCIZ "Codemaker is: "
    // Store block of memory of 128 bytes to store the string
    codeMakerMsg: .BLOCK 128
    // Display whoIsCodeMaker Query prompt:
    whoIsCodeBreakerMsg: .ASCIZ "\nCodebreaker is: "
    // Store block of memory of 128 bytes to store the string
    codeBreakerMsg: .BLOCK 128
    // Display guessLimit Query prompt:
    whatIsGuessLimitMsg: .ASCIZ "\nGuess Limit: "
```



```
// Set codeMaker value to R4
    MOV R4, #codeMaker
// Take input from user and store to R4
    STR R4, .ReadString
// Print codeMaker value from R4
    STR R4, .WriteString
// Prompt and store CodeBreaker Name
// Set whoIsCodeMaker Query prompt to R10
    MOV R10, #whoIsCodeBreaker
// print whoIsCodeMaker Query from R4
    STR R10, .WriteString
// Set codeMaker value to R5
    MOV R5, #codeBreaker
// Take input from user and store to R5
    STR R5, .ReadString
// Print codeMaker value from R5
    STR R5, .WriteString
// Prompt and store GuessLimit for the session
// Set guessLimit Query prompt to R10
    MOV R10, #whatIsGuessLimit
// Print whatIsGuessLimit from R10
    STR R10, .WriteString
// Take input from user and store to R11
    LDR R11, .InputNum
// Print guessLimit from R11
    STR R11, .WriteSignedNum
// Stop Program.
    HALT
// Program functions:
// Display whoIsCodeMaker Query prompt:
whoIsCodeMaker: .ASCIZ "Codemaker is: "
// Store block of memory of 128 bytes to store the string
codeMaker: .BLOCK 128
// Display whoIsCodeMaker Query prompt:
whoIsCodeBreaker: .ASCIZ "\nCodebreaker is: "
// Store block of memory of 128 bytes to store the string
codeBreaker: .BLOCK 128
// Display guessLimit Query prompt:
whatIsGuessLimit: .ASCIZ "\nGuess Limit: "
```

```
PC  0x0000003c
LR  0x00000000
SP  0x00100000
R12 0x00000000
R11 0x0000000c
R10 0x0000015d
R9  0x00000000
R8  0x00000000
R7  0x00000000
R6  0x00000000
R5  0x000000dd
R4  0x0000004b
R3  0x00000000
R2  0x00000000
R1  0x00000000
R0  0x00000000
```

Count 15

Current Instruction

Status bits NZCV 0000

```
Codemaker is: Luke
Codebreaker is: Vandy
Guess Limit: 12
Program HALTED. STOP, LOAD or EDIT
```

```
12
```

```
000           0x0        0x4        0x8        0xc
0x0000 0xe3a0a03c 0xe50fa0ec 0xe3a0404b 0xe50f40b4
0x0001 0xe50f40f8 0xe3a0a0cb 0xe50fa100 0xe3a050dd
0x0002 0xe50f50c8 0xe50f510c 0xe300a15d 0xe50fa114
0x0003 0xe51fb130 0xe50fb12c 0xe1000070 0x65646f43
0x0004 0x656b616d 0x73692072 0x4c00203a 0x00656b75
0x0005 0x00000000 0x00000000 0x00000000 0x00000000
0x0006 0x00000000 0x00000000 0x00000000 0x00000000
0x0007 0x00000000 0x00000000 0x00000000 0x00000000
0x0008 0x00000000 0x00000000 0x00000000 0x00000000
0x0009 0x00000000 0x00000000 0x00000000 0x00000000
0x000a 0x00000000 0x00000000 0x00000000 0x00000000
0x000b 0x00000000 0x00000000 0x00000000 0x00000000
0x000c 0x00000000 0x00000000 0x0a000000 0x65646f43
0x000d 0x61657262 0x2072656b 0x203a7369 0x6e615600
0x000e 0x00007964 0x00000000 0x00000000 0x00000000
0x000f 0x00000000 0x00000000 0x00000000 0x00000000
0x0010 0x00000000 0x00000000 0x00000000 0x00000000
0x0011 0x00000000 0x00000000 0x00000000 0x00000000
0x0012 0x00000000 0x00000000 0x00000000 0x00000000
0x0013 0x00000000 0x00000000 0x00000000 0x00000000
0x0014 0x00000000 0x00000000 0x00000000 0x00000000
0x0015 0x00000000 0x00000000 0x00000000 0x75470a00
0x0016 0x20737365 0x696d694c 0x00203a74 0x00000000
0x0017 0x00000000 0x00000000 0x00000000 0x00000000
0x0018 0x00000000 0x00000000 0x00000000 0x00000000
0x0019 0x00000000 0x00000000 0x00000000 0x00000000
0x001a 0x00000000 0x00000000 0x00000000 0x00000000
0x001b 0x00000000 0x00000000 0x00000000 0x00000000
0x001c 0x00000000 0x00000000 0x00000000 0x00000000
0x001d 0x00000000 0x00000000 0x00000000 0x00000000
0x001e 0x00000000 0x00000000 0x00000000 0x00000000
0x001f 0x00000000 0x00000000 0x00000000 0x00000000
```

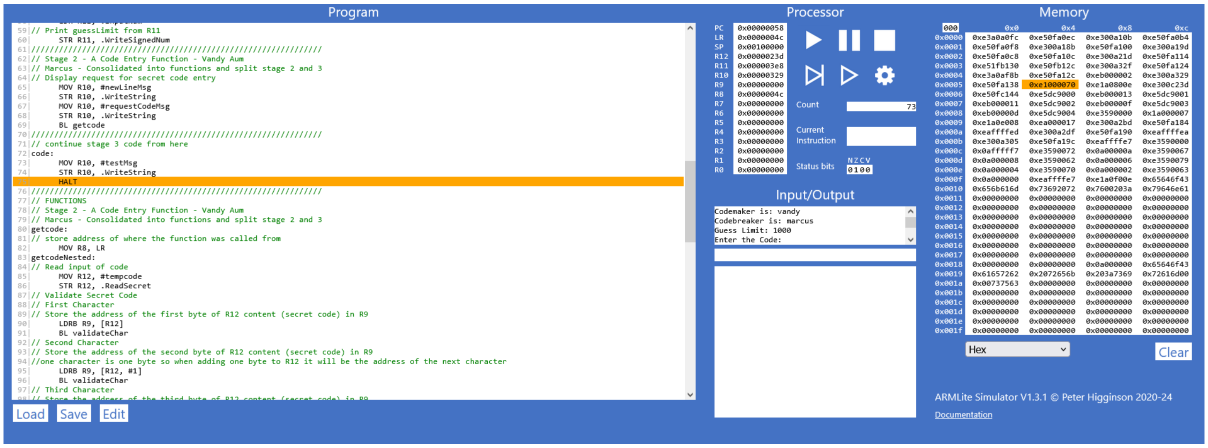
Hex

ARMLite Simulator V1.3.1 © Peter Higginson 2020-24
Documentation

Figure 1: Stage 1: Functional Screenshot

Stage 2 (stage2.txt)

In stage 2 a function `getcode` was created to receive input of a code and validate that it follows the rules of the game. After receiving input, the value of each character is extracted from the string using `LDRB` before branching to `validateChar` where it is checked against all valid characters. The fifth character of the string is then checked and returns an error if it has any value.



```
// Print guessLimit from R11
      STR R11, .WriteSignedNum
//////////////////////////////////////////////////////////////
// Stage 2 - A Code Entry Function - Vandy Aum
// Marcus - Consolidated into functions and split stage 2 and 3
// Display request for secret code entry
      MOV R10, #newLineMsg
      STR R10, .WriteString
      MOV R10, #requestCodeMsg
      STR R10, .WriteString
      BL getcode
//////////////////////////////////////////////////////////////
// continue stage 3 code from here
code:
      MOV R10, #testMsg
      STR R10, .WriteString
      HALT
//////////////////////////////////////////////////////////////
// FUNCTIONS
// Stage 2 - A Code Entry Function - Vandy Aum
// Marcus - Consolidated into functions and split stage 2 and 3
getcode:
// store address of where the function was called from
      MOV R8, LR
getcodeNested:
// Read input of code
      MOV R12, #tempcode
      STR R12, .ReadSecret
// Validate Secret Code
// First Character
// Store the address of the first byte of R12 content (secret code) in R9
      LDRB R9, [R12]
      BL validateChar
// Second Character
// Store the address of the second byte of R12 content (secret code) in R9
//one character is one byte so when adding one byte to R12 it will be the address of the next character
      LDRB R9, [R12, #1]
      BL validateChar
// Third Character
// Store the address of the third byte of R12 content (secret code) in R9
```

```
PC   0x00000058
LR   0x0000004c
SP   0x00100000
R12  0x0000023d
R11  0x000003e8
R10  0x00000329
R9   0x00000000
R8   0x0000004c
R7   0x00000000
R6   0x00000000
R5   0x00000000
R4   0x00000000
R3   0x00000000
R2   0x00000000
R1   0x00000000
R0   0x00000000

Count 73
Status bits NZCV 0100
```

```
Codemaker is: vandy
Codebreaker is: marcus
Guess Limit: 1000
Enter the Code:
```

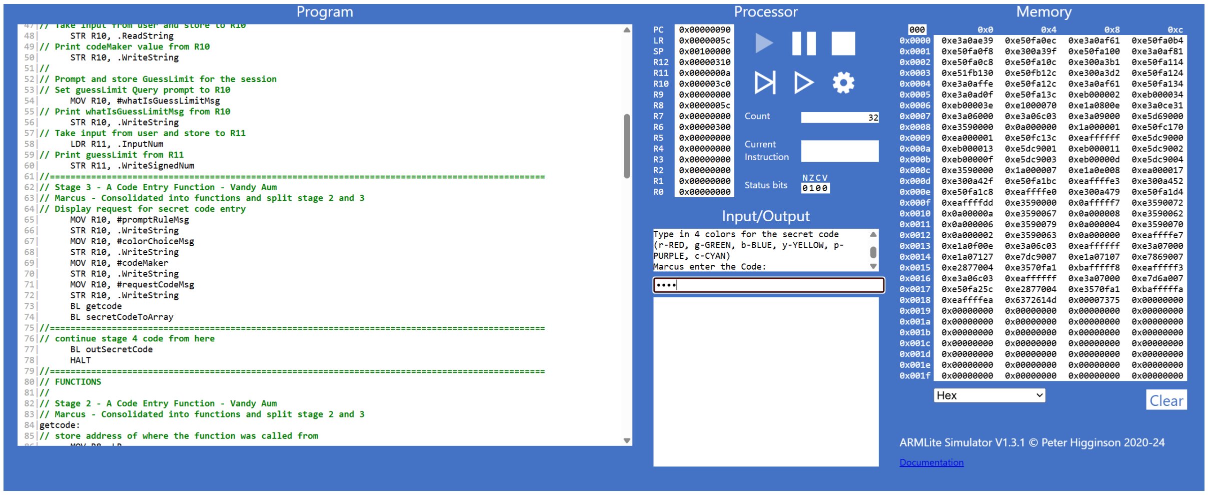
```
000           0x0         0x4         0x8         0xc
0x0000  0xe3a0a0fc  0xe50fa0ec  0xe300a10b  0xe50fa0b4
0x0001  0xe50fa0f8  0xe300a18b  0xe50fa100  0xe300a19d
0x0002  0xe50fa0c8  0xe50fa10c  0xe300a21d  0xe50fa114
0x0003  0xe51fb130  0xe50fb12c  0xe300a32f  0xe50fa124
0x0004  0xe3a0af8b  0xe50fa12c  0xeb000002  0xe300a329
0x0005  0xe50fa138  0xe1000070  0xe1a0800e  0xe300c23d
0x0006  0xe50fc144  0xe5dc9000  0xeb000013  0xe5dc9001
0x0007  0xeb000011  0xe5dc9002  0xeb00000f  0xe5dc9003
0x0008  0xeb00000d  0xe5dc9004  0xe3590000  0x1a000007
0x0009  0xe1a0e008  0xea000017  0xe300a2bd  0xe50fa184
0x000a  0xeaffffed  0xe300a2df  0xe50fa190  0xeaffffea
0x000b  0xe300a305  0xe50fa19c  0xeaffffe7  0xe3590000
0x000c  0x0afffff7  0xe3590072  0x0a00000a  0xe3590067
0x000d  0x0a000008  0xe3590062  0x0a000006  0xe3590079
0x000e  0x0a000004  0xe3590070  0x0a000002  0xe3590063
0x000f  0x0a000000  0xeaffffe7  0xe1a0f00e  0x65646f43
0x0010  0x656b616d  0x73692072  0x7600203a  0x79646e61
0x0011  0x00000000  0x00000000  0x00000000  0x00000000
0x0012  0x00000000  0x00000000  0x00000000  0x00000000
0x0013  0x00000000  0x00000000  0x00000000  0x00000000
0x0014  0x00000000  0x00000000  0x00000000  0x00000000
0x0015  0x00000000  0x00000000  0x00000000  0x00000000
0x0016  0x00000000  0x00000000  0x00000000  0x00000000
0x0017  0x00000000  0x00000000  0x00000000  0x00000000
0x0018  0x00000000  0x00000000  0x0a000000  0x65646f43
0x0019  0x61657262  0x2072656b  0x203a7369  0x72616d00
0x001a  0x00737563  0x00000000  0x00000000  0x00000000
0x001b  0x00000000  0x00000000  0x00000000  0x00000000
0x001c  0x00000000  0x00000000  0x00000000  0x00000000
0x001d  0x00000000  0x00000000  0x00000000  0x00000000
0x001e  0x00000000  0x00000000  0x00000000  0x00000000
0x001f  0x00000000  0x00000000  0x00000000  0x00000000
```

ARMLite Simulator V1.3.1 © Peter Higginson 2020-24

Figure 2: Stage 2: Functional Screenshot

Stage 3 (stage3.txt)

In stage 3 the 'codeToArray' function was created to convert the string 'tempcode' into an array. The `getcode` function was also modified to utilize `.ReadSecret` the first time it runs (always the code maker's turn) to hide the entered code from the code breaker.



```
// Take input from user and store to R10
      STR R10, .ReadString
// Print codeMaker value from R10
      STR R10, .WriteString
//
// Prompt and store GuessLimit for the session
// Set guessLimit Query prompt to R10
      MOV R10, #whatIsGuessLimitMsg
// Print whatIsGuessLimitMsg from R10
      STR R10, .WriteString
// Take input from user and store to R11
      LDR R11, .InputNum
// Print guessLimit from R11
      STR R11, .WriteSignedNum
//==========================================================================================
// Stage 3 - A Code Entry Function - Vandy Aum
// Marcus - Consolidated into functions and split stage 2 and 3
// Display request for secret code entry
      MOV R10, #promptRuleMsg
      STR R10, .WriteString
      MOV R10, #colorChoiceMsg
      STR R10, .WriteString
      MOV R10, #codeMaker
      STR R10, .WriteString
      MOV R10, #requestCodeMsg
      STR R10, .WriteString
      BL getcode
      BL secretCodeToArray
//==========================================================================================
// continue stage 4 code from here
      BL outSecretCode
      HALT
//==========================================================================================
// FUNCTIONS
//
// Stage 2 - A Code Entry Function - Vandy Aum
// Marcus - Consolidated into functions and split stage 2 and 3
getcode:
// store address of where the function was called from
      MOV R8, LR
```

```
PC   0x00000090
LR   0x0000005c
SP   0x00100000
R12  0x00000310
R11  0x0000000a
R10  0x000003c0
R9   0x00000000
R8   0x0000005c
R7   0x00000000
R6   0x00000300
R5   0x00000000
R4   0x00000000
R3   0x00000000
R2   0x00000000
R1   0x00000000
R0   0x00000000

Count 32
Status bits NZCV 0100
```

```
Type in 4 colors for the secret code
(r-RED, g-GREEN, b-BLUE, y-YELLOW, p-
PURPLE, c-CYAN)
Marcus enter the Code:
••••
```

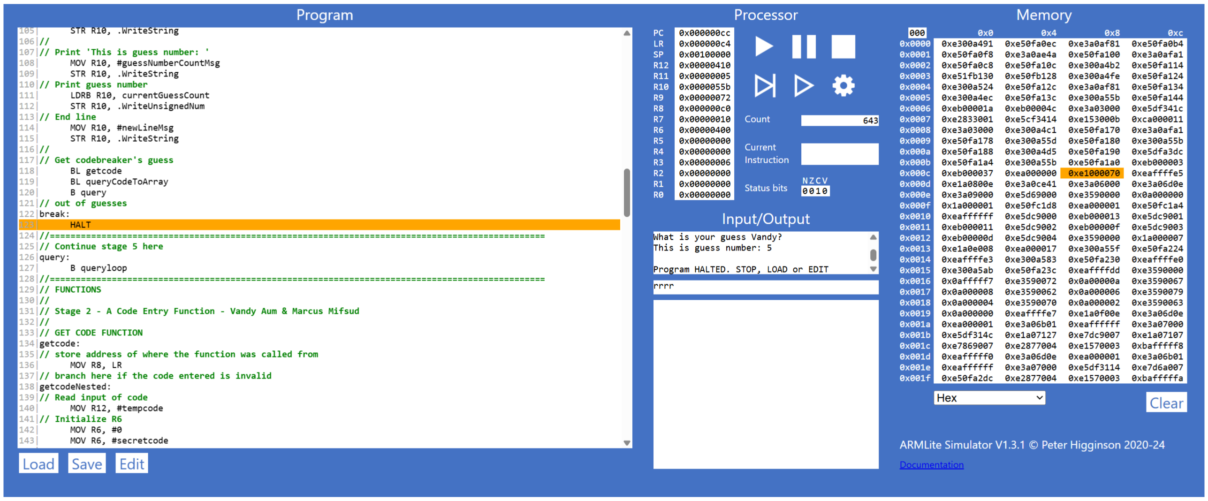
```
000           0x0         0x4         0x8         0xc
0x0000  0xe3a0ae39  0xe50fa0ec  0xe3a0af61  0xe50fa0b4
0x0001  0xe50fa0f8  0xe300a39f  0xe50fa100  0xe3a0af81
0x0002  0xe50fa0c8  0xe50fa10c  0xe300a3b1  0xe50fa114
0x0003  0xe51fb130  0xe50fb12c  0xe300a3d2  0xe50fa124
0x0004  0xe3a0affe  0xe50fa12c  0xe3a0af61  0xe50fa134
0x0005  0xe3a0ad0f  0xe50fa13c  0xeb000002  0xeb000034
0x0006  0xeb00003e  0xe1000070  0xe1a0800e  0xe3a0ce31
0x0007  0xe3a06000  0xe3a06c03  0xe3a09000  0xe5d69000
0x0008  0xe3590000  0x0a000000  0x1a000001  0xe50fc170
0x0009  0xea000001  0xe50fc13c  0xeaffffff  0xe5dc9000
0x000a  0xeb000013  0xe5dc9001  0xeb000011  0xe5dc9002
0x000b  0xeb00000f  0xe5dc9003  0xeb00000d  0xe5dc9004
0x000c  0xe3590000  0x1a000007  0xe1a0e008  0xea000017
0x000d  0xe300a42f  0xe50fa1bc  0xeaffffe3  0xe300a452
0x000e  0xe50fa1c8  0xeaffffe0  0xe300a479  0xe50fa1d4
0x000f  0xeaffffdd  0xe3590000  0x0afffff7  0xe3590072
0x0010  0x0a00000a  0xe3590067  0x0a000008  0xe3590062
0x0011  0x0a000006  0xe3590079  0x0a000004  0xe3590070
0x0012  0x0a000002  0xe3590063  0x0a000000  0xeaffffe7
0x0013  0xe1a0f00e  0xe3a06c03  0xeaffffff  0xe3a07000
0x0014  0xe1a07127  0xe7dc9007  0xe1a07107  0xe7869007
0x0015  0xe2877004  0xe3570fa1  0xbafffff8  0xeafffff3
0x0016  0xe3a06c03  0xeaffffff  0xe3a07000  0xe7d6a007
0x0017  0xe50fa25c  0xe2877004  0xe3570fa1  0xbafffffa
0x0018  0xeaffffea  0x6372614d  0x00007375  0x00000000
0x0019  0x00000000  0x00000000  0x00000000  0x00000000
0x001a  0x00000000  0x00000000  0x00000000  0x00000000
0x001b  0x00000000  0x00000000  0x00000000  0x00000000
0x001c  0x00000000  0x00000000  0x00000000  0x00000000
0x001d  0x00000000  0x00000000  0x00000000  0x00000000
0x001e  0x00000000  0x00000000  0x00000000  0x00000000
0x001f  0x00000000  0x00000000  0x00000000  0x00000000
```

ARMLite Simulator V1.3.1 © Peter Higginson 2020-24

Figure 3: Stage 3: Screenshot showing code maker code entry

Stage 4 (stage4.txt)

In stage 4 the `queryloop` function was created which increments the guess counter before checking if the code breaker has exceeded the guess limit. If not, the code breaker is requested to enter their guess using the `getcode` function. The code then branches back to the start of `queryloop` and continues looping until the guess limit is met.



```
        STR R10, .WriteString
//
// Print 'This is guess number: '
        MOV R10, #guessNumberCountMsg
        STR R10, .WriteString
// Print guess number
        LDRB R10, currentGuessCount
        STR R10, .WriteUnsignedNum
// End line
        MOV R10, #newLineMsg
        STR R10, .WriteString
//
// Get codebreaker's guess
        BL getcode
        BL queryCodeToArray
        B query
// out of guesses
break:
        HALT
//==========================================================================================
// Continue stage 5 here
query:
        B queryloop
//==========================================================================================
// FUNCTIONS
//
// Stage 2 - A Code Entry Function - Vandy Aum & Marcus Mifsud
//
// GET CODE FUNCTION
getcode:
// store address of where the function was called from
        MOV R8, LR
// branch here if the code entered is invalid
getcodeNested:
// Read input of code
        MOV R12, #tempcode
// Initialize R6
        MOV R6, #0
        MOV R6, #secretcode
```

```
PC   0x000000cc
LR   0x000000c4
SP   0x00100000
R12  0x00000410
R11  0x00000005
R10  0x0000055b
R9   0x00000072
R8   0x000000c0
R7   0x00000010
R6   0x00000400
R5   0x00000000
R4   0x00000000
R3   0x00000006
R2   0x00000000
R1   0x00000000
R0   0x00000000

Count                643
Current Instruction
Status bits          NZCV
                     0010
```

```
What is your guess Vandy?
This is guess number: 5

Program HALTED. STOP, LOAD or EDIT
```

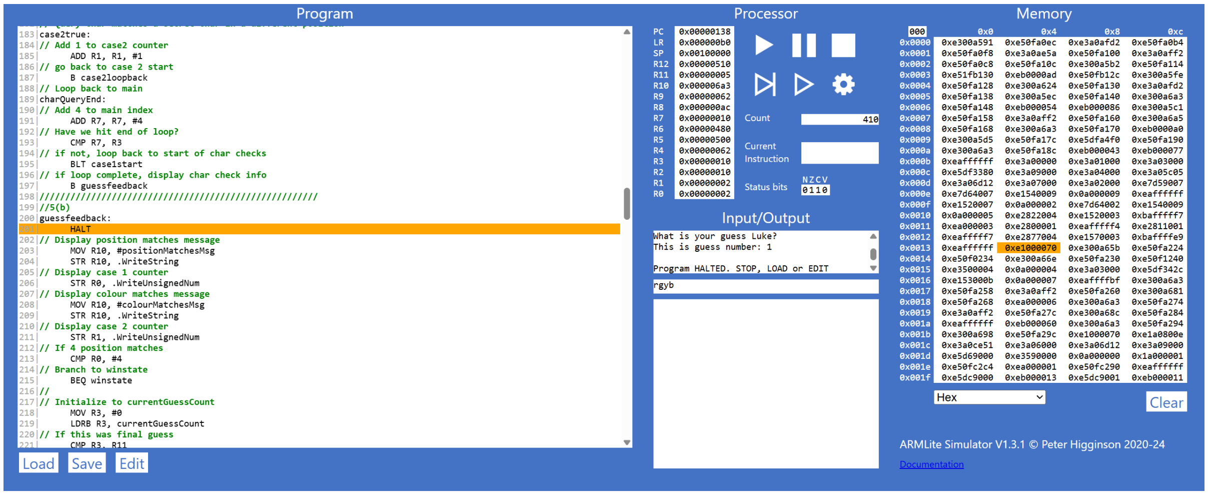
```
rrrr
```

```
   000          0x0         0x4         0x8         0xc
0x0000  0xe300a491  0xe50fa0ec  0xe3a0af81  0xe50fa0b4
0x0001  0xe50fa0f8  0xe3a0ae4a  0xe50fa100  0xe3a0afa1
0x0002  0xe50fa0c8  0xe50fa10c  0xe300a4b2  0xe50fa114
0x0003  0xe51fb130  0xe50fb128  0xe300a4fe  0xe50fa124
0x0004  0xe300a524  0xe50fa12c  0xe3a0af81  0xe50fa134
0x0005  0xe300a4ec  0xe50fa13c  0xe300a55b  0xe50fa144
0x0006  0xeb00001a  0xeb00004c  0xe3a03000  0xe5df341c
0x0007  0xe2833001  0xe5cf3414  0xe153000b  0xca000011
0x0008  0xe3a03000  0xe300a4c1  0xe50fa170  0xe3a0afa1
0x0009  0xe50fa178  0xe300a55d  0xe50fa180  0xe300a55b
0x000a  0xe50fa188  0xe300a4d5  0xe50fa190  0xe5dfa3dc
0x000b  0xe50fa1a4  0xe300a55b  0xe50fa1a0  0xeb000003
0x000c  0xeb000037  0xea000000  0xe1000070  0xeaffffe5
0x000d  0xe1a0800e  0xe3a0ce41  0xe3a06000  0xe3a06d0e
0x000e  0xe3a09000  0xe5d69000  0xe3590000  0x0a000000
0x000f  0x1a000001  0xe50fc1d8  0xea000001  0xe50fc1a4
0x0010  0xeaffffff  0xe5dc9000  0xeb000013  0xe5dc9001
0x0011  0xeb000011  0xe5dc9002  0xeb00000f  0xe5dc9003
0x0012  0xeb00000d  0xe5dc9004  0xe3590000  0x1a000007
0x0013  0xe1a0e008  0xea000017  0xe300a55f  0xe50fa224
0x0014  0xeaffffe3  0xe300a583  0xe50fa230  0xeaffffe0
0x0015  0xe300a5ab  0xe50fa23c  0xeaffffdd  0xe3590000
0x0016  0x0afffff7  0xe3590072  0x0a00000a  0xe3590067
0x0017  0x0a000008  0xe3590062  0x0a000006  0xe3590079
0x0018  0x0a000004  0xe3590070  0x0a000002  0xe3590063
0x0019  0x0a000000  0xeaffffe7  0xe1a0f00e  0xe3a06d0e
0x001a  0xea000001  0xe3a06b01  0xeaffffff  0xe3a07000
0x001b  0xe5df314c  0xe1a07127  0xe7dc9007  0xe1a07107
0x001c  0xe7869007  0xe2877004  0xe1570003  0xbafffff8
0x001d  0xeafffff0  0xe3a06d0e  0xea000001  0xe3a06b01
0x001e  0xeaffffff  0xe3a07000  0xe5df3114  0xe7d6a007
0x001f  0xe50fa2dc  0xe2877004  0xe1570003  0xbafffffa
```

Figure 4: Stage 4: Functional Screenshot

Stage 5a (stage5a.txt)

In stage 5 the `comparecodes` function was created, it utilizes a main loop for each character of the query code and a nested loop for each character of the secret code testing for case 2.



```
case2true:
// Add 1 to case2 counter
        ADD R1, R1, #1
// go back to case 2 start
        B case2loopback
// Loop back to main
charQueryEnd:
// Add 4 to main index
        ADD R7, R7, #4
// Have we hit end of loop?
        CMP R7, R3
// if not, loop back to start of char checks
        BLT case1start
// if loop complete, display char check info
        B guessfeedback
//////////////////////////////////////////////////////
//5(b)
guessfeedback:
        HALT
// Display position matches message
        MOV R10, #positionMatchesMsg
        STR R10, .WriteString
// Display case 1 counter
        STR R0, .WriteUnsignedNum
// Display colour matches message
        MOV R10, #colourMatchesMsg
        STR R10, .WriteString
// Display case 2 counter
        STR R1, .WriteUnsignedNum
// If 4 position matches
        CMP R0, #4
// Branch to winstate
        BEQ winstate
//
// Initialize to currentGuessCount
        MOV R3, #0
        LDRB R3, currentGuessCount
// If this was final guess
        CMP R3, R11
```

```
PC   0x00000138
LR   0x000000b0
SP   0x00100000
R12  0x00000510
R11  0x00000005
R10  0x000006a3
R9   0x00000062
R8   0x000000ac
R7   0x00000010
R6   0x00000480
R5   0x00000500
R4   0x00000062
R3   0x00000010
R2   0x00000010
R1   0x00000002
R0   0x00000002

Count                410
Current Instruction
Status bits          NZCV
                     0110
```

```
What is your guess Luke?
This is guess number: 1

Program HALTED. STOP, LOAD or EDIT
```

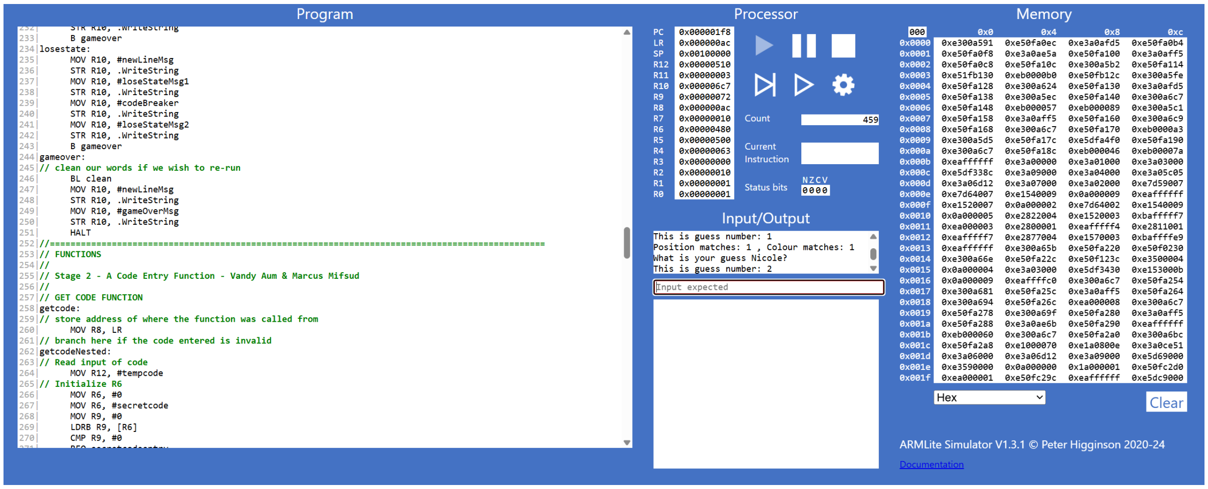
```
rgyb
```

```
   000          0x0         0x4         0x8         0xc
0x0000  0xe300a591  0xe50fa0ec  0xe3a0afd2  0xe50fa0b4
0x0001  0xe50fa0f8  0xe3a0ae5a  0xe50fa100  0xe3a0aff2
0x0002  0xe50fa0c8  0xe50fa10c  0xe300a5b2  0xe50fa114
0x0003  0xe51fb130  0xeb0000ad  0xe50fb12c  0xe300a5fe
0x0004  0xe50fa128  0xe300a624  0xe50fa130  0xe3a0afd2
0x0005  0xe50fa138  0xe300a5ec  0xe50fa140  0xe300a6a3
0x0006  0xe50fa148  0xeb000054  0xeb000086  0xe300a5c1
0x0007  0xe50fa158  0xe3a0aff2  0xe50fa160  0xe300a6a5
0x0008  0xe50fa168  0xe300a6a3  0xe50fa170  0xeb0000a0
0x0009  0xe300a5d5  0xe50fa17c  0xe5dfa4f0  0xe50fa190
0x000a  0xe300a6a3  0xe50fa18c  0xeb000043  0xeb000077
0x000b  0xeaffffff  0xe3a00000  0xe3a01000  0xe3a03000
0x000c  0xe5df3380  0xe3a09000  0xe3a04000  0xe3a05c05
0x000d  0xe3a06d12  0xe3a07000  0xe3a02000  0xe7d59007
0x000e  0xe7d64007  0xe1540009  0x0a000009  0xeaffffff
0x000f  0xe1520007  0x0a000002  0xe7d64002  0xe1540009
0x0010  0x0a000005  0xe2822004  0xe1520003  0xbafffff7
0x0011  0xea000003  0xe2800001  0xeafffff4  0xe2811001
0x0012  0xeafffff7  0xe2877004  0xe1570003  0xbaffffe9
0x0013  0xeaffffff  0xe1000070  0xe300a65b  0xe50fa224
0x0014  0xe50f0234  0xe300a66e  0xe50fa230  0xe50f1240
0x0015  0xe3500004  0x0a000004  0xe3a03000  0xe5df342c
0x0016  0xe153000b  0x0a000007  0xeaffffbf  0xe300a6a3
0x0017  0xe50fa258  0xe3a0aff2  0xe50fa260  0xe300a681
0x0018  0xe50fa268  0xea000006  0xe300a6a3  0xe50fa274
0x0019  0xe3a0aff2  0xe50fa27c  0xe300a68c  0xe50fa284
0x001a  0xeaffffff  0xeb000060  0xe300a6a3  0xe50fa294
0x001b  0xe300a698  0xe50fa29c  0xe1000070  0xe1a0800e
0x001c  0xe3a0ce51  0xe3a06000  0xe3a06d12  0xe3a09000
0x001d  0xe5d69000  0xe3590000  0x0a000000  0x1a000001
0x001e  0xe50fc2c4  0xea000001  0xe50fc290  0xeaffffff
0x001f  0xe5dc9000  0xeb000013  0xe5dc9001  0xeb000011
```

Figure 5: Stage 5a: Screenshot showing 2 exact matches and 2 colour matches (stored in R0 & R1)

Stage 5b (stage5b.txt)

In stage 5 the guessfeedback function was created which displays the result of comparecodes. If the result of case 1 is 4 (the codes fully match) the code branches to winstate which display's a win message and then branches to gameover which ends the game. The logic for incrementing the current guess count and checking the guess limit was also moved from the start of queryloop to the end of guessfeedback and branches to losestate if the guess limit has been exceeded without a full code match. If the code breaker neither wins or loses at this point, the code loops back to the start of queryloop to allow another guess.



```
        STR R10, .WriteString
        B gameover
losestate:
        MOV R10, #newLineMsg
        STR R10, .WriteString
        MOV R10, #loseStateMsg1
        STR R10, .WriteString
        MOV R10, #codeBreaker
        STR R10, .WriteString
        MOV R10, #loseStateMsg2
        STR R10, .WriteString
        B gameover
gameover:
// clean our words if we wish to re-run
        BL clean
        MOV R10, #newLineMsg
        STR R10, .WriteString
        MOV R10, #gameOverMsg
        STR R10, .WriteString
        HALT
//==========================================================================================
// FUNCTIONS
//
// Stage 2 - A Code Entry Function - Vandy Aum & Marcus Mifsud
//
// GET CODE FUNCTION
getcode:
// store address of where the function was called from
        MOV R8, LR
// branch here if the code entered is invalid
getcodeNested:
// Read input of code
        MOV R12, #tempcode
// Initialize R6
        MOV R6, #0
        MOV R6, #secretcode
        MOV R9, #0
        LDRB R9, [R6]
        CMP R9, #0
```

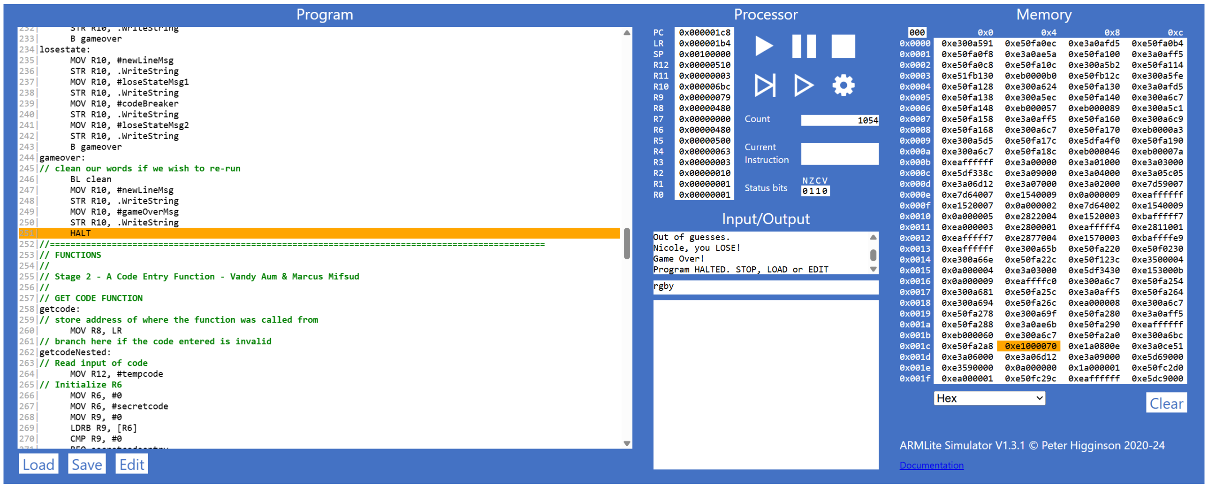
```
PC   0x000001f8
LR   0x000000ac
SP   0x00100000
R12  0x00000510
R11  0x00000003
R10  0x000006c7
R9   0x00000072
R8   0x000000ac
R7   0x00000010
R6   0x00000480
R5   0x00000500
R4   0x00000063
R3   0x00000000
R2   0x00000010
R1   0x00000001
R0   0x00000001

Count  459
Current Instruction
Status bits  NZCV
             0000
```

```
This is guess number: 1
Position matches: 1 , Colour matches: 1
What is your guess Nicole?
This is guess number: 2
```

```
Input expected
```

```
000           0x0         0x4         0x8         0xc
0x0000  0xe300a591  0xe50fa0ec  0xe3a0afd5  0xe50fa0b4
0x0001  0xe50fa0f8  0xe3a0ae5a  0xe50fa100  0xe3a0aff5
0x0002  0xe50fa0c8  0xe50fa10c  0xe300a5b2  0xe50fa114
0x0003  0xe51fb130  0xeb0000b0  0xe50fb12c  0xe300a5fe
0x0004  0xe50fa128  0xe300a624  0xe50fa130  0xe3a0afd5
0x0005  0xe50fa138  0xe300a5ec  0xe50fa140  0xe300a6c7
0x0006  0xe50fa148  0xeb000057  0xeb000089  0xe300a5c1
0x0007  0xe50fa158  0xe3a0aff5  0xe50fa160  0xe300a6c9
0x0008  0xe50fa168  0xe300a6c7  0xe50fa170  0xeb0000a3
0x0009  0xe300a5d5  0xe50fa17c  0xe5dfa4f0  0xe50fa190
0x000a  0xe300a6c7  0xe50fa18c  0xeb000046  0xeb00007a
0x000b  0xeaffffff  0xe3a00000  0xe3a01000  0xe3a03000
0x000c  0xe5df338c  0xe3a09000  0xe3a04000  0xe3a05c05
0x000d  0xe3a06d12  0xe3a07000  0xe3a02000  0xe7d59007
0x000e  0xe7d64007  0xe1540009  0x0a000009  0xeaffffff
0x000f  0xe1520007  0x0a000002  0xe7d64002  0xe1540009
0x0010  0x0a000005  0xe2822004  0xe1520003  0xbafffff7
0x0011  0xea000003  0xe2800001  0xeafffff4  0xe2811001
0x0012  0xeafffff7  0xe2877004  0xe1570003  0xbaffffe9
0x0013  0xeaffffff  0xe300a65b  0xe50fa220  0xe50f0230
0x0014  0xe300a66e  0xe50fa22c  0xe50f123c  0xe3500004
0x0015  0x0a000004  0xe3a03000  0xe5df3430  0xe153000b
0x0016  0x0a000009  0xeaffffc0  0xe300a6c7  0xe50fa254
0x0017  0xe300a681  0xe50fa25c  0xe3a0aff5  0xe50fa264
0x0018  0xe300a694  0xe50fa26c  0xea000008  0xe300a6c7
0x0019  0xe50fa278  0xe300a69f  0xe50fa280  0xe3a0aff5
0x001a  0xe50fa288  0xe3a0ae6b  0xe50fa290  0xeaffffff
0x001b  0xeb000060  0xe300a6c7  0xe50fa2a0  0xe300a6bc
0x001c  0xe50fa2a8  0xe1000070  0xe1a0800e  0xe3a0ce51
0x001d  0xe3a06000  0xe3a06d12  0xe3a09000  0xe5d69000
0x001e  0xe3590000  0x0a000000  0x1a000001  0xe50fc2d0
0x001f  0xea000001  0xe50fc29c  0xeaffffff  0xe5dc9000
```

Figure 6: Stage 5b: Screenshot showing feedback for a guess



```
        STR R10, .WriteString
        B gameover
losestate:
        MOV R10, #newLineMsg
        STR R10, .WriteString
        MOV R10, #loseStateMsg1
        STR R10, .WriteString
        MOV R10, #codeBreaker
        STR R10, .WriteString
        MOV R10, #loseStateMsg2
        STR R10, .WriteString
        B gameover
gameover:
// clean our words if we wish to re-run
        BL clean
        MOV R10, #newLineMsg
        STR R10, .WriteString
        MOV R10, #gameOverMsg
        STR R10, .WriteString
        HALT
//==========================================================================================
// FUNCTIONS
//
// Stage 2 - A Code Entry Function - Vandy Aum & Marcus Mifsud
//
// GET CODE FUNCTION
getcode:
// store address of where the function was called from
        MOV R8, LR
// branch here if the code entered is invalid
getcodeNested:
// Read input of code
        MOV R12, #tempcode
// Initialize R6
        MOV R6, #0
        MOV R6, #secretcode
        MOV R9, #0
        LDRB R9, [R6]
        CMP R9, #0
```

```
PC   0x000001c8
LR   0x000001b4
SP   0x00100000
R12  0x00000510
R11  0x00000003
R10  0x000006bc
R9   0x00000079
R8   0x00000480
R7   0x00000000
R6   0x00000480
R5   0x00000500
R4   0x00000063
R3   0x00000003
R2   0x00000010
R1   0x00000001
R0   0x00000001

Count  1054
Current Instruction
Status bits  NZCV
             0110
```

```
Out of guesses.
Nicole, you LOSE!
Game Over!
Program HALTED. STOP, LOAD or EDIT
```

```
rgby
```

```
000           0x0         0x4         0x8         0xc
0x0000  0xe300a591  0xe50fa0ec  0xe3a0afd5  0xe50fa0b4
0x0001  0xe50fa0f8  0xe3a0ae5a  0xe50fa100  0xe3a0aff5
0x0002  0xe50fa0c8  0xe50fa10c  0xe300a5b2  0xe50fa114
0x0003  0xe51fb130  0xeb0000b0  0xe50fb12c  0xe300a5fe
0x0004  0xe50fa128  0xe300a624  0xe50fa130  0xe3a0afd5
0x0005  0xe50fa138  0xe300a5ec  0xe50fa140  0xe300a6c7
0x0006  0xe50fa148  0xeb000057  0xeb000089  0xe300a5c1
0x0007  0xe50fa158  0xe3a0aff5  0xe50fa160  0xe300a6c9
0x0008  0xe50fa168  0xe300a6c7  0xe50fa170  0xeb0000a3
0x0009  0xe300a5d5  0xe50fa17c  0xe5dfa4f0  0xe50fa190
0x000a  0xe300a6c7  0xe50fa18c  0xeb000046  0xeb00007a
0x000b  0xeaffffff  0xe3a00000  0xe3a01000  0xe3a03000
0x000c  0xe5df338c  0xe3a09000  0xe3a04000  0xe3a05c05
0x000d  0xe3a06d12  0xe3a07000  0xe3a02000  0xe7d59007
0x000e  0xe7d64007  0xe1540009  0x0a000009  0xeaffffff
0x000f  0xe1520007  0x0a000002  0xe7d64002  0xe1540009
0x0010  0x0a000005  0xe2822004  0xe1520003  0xbafffff7
0x0011  0xea000003  0xe2800001  0xeafffff4  0xe2811001
0x0012  0xeafffff7  0xe2877004  0xe1570003  0xbaffffe9
0x0013  0xeaffffff  0xe300a65b  0xe50fa220  0xe50f0230
0x0014  0xe300a66e  0xe50fa22c  0xe50f123c  0xe3500004
0x0015  0x0a000004  0xe3a03000  0xe5df3430  0xe153000b
0x0016  0x0a000009  0xeaffffc0  0xe300a6c7  0xe50fa254
0x0017  0xe300a681  0xe50fa25c  0xe3a0aff5  0xe50fa264
0x0018  0xe300a694  0xe50fa26c  0xea000008  0xe300a6c7
0x0019  0xe50fa278  0xe300a69f  0xe50fa280  0xe3a0aff5
0x001a  0xe50fa288  0xe3a0ae6b  0xe50fa290  0xeaffffff
0x001b  0xeb000060  0xe300a6c7  0xe50fa2a0  0xe300a6bc
0x001c  0xe50fa2a8  0xe1000070  0xe1a0800e  0xe3a0ce51
0x001d  0xe3a06000  0xe3a06d12  0xe3a09000  0xe5d69000
0x001e  0xe3590000  0x0a000000  0x1a000001  0xe50fc2d0
0x001f  0xea000001  0xe50fc29c  0xeaffffff  0xe5dc9000
```

Figure 7: Stage 5b: Screenshot showing lose state

Assumptions

No restrictions for user submitted Guess Limit

Reasonable number of guesses will be submitted as input for the user without controls. The application does not constrict the user-entry value of the number of guesses to either a numerical entry limit, nor a theoretical mathematical limit of guesses needed to get the right answer. For example, as per the rules of Mastermind, the total sequences available to guess from is expressed by:

$$\text{Total Sequences} = \text{Number of options}^{\text{Number of places}}$$

$$\text{Total Sequences} = 6^4 = 1296$$

No Duplicate Guess controls

There are no validation checks for duplicate sequence submissions made by the user. This means that the user is burning an opportunity to guess within the specified limit, but also means that they have increased the number of guesses that could potentially be needed to obtain the correct outcome if there was no limit specified. That is, for each duplicate guess d, the number of total sequences increases by 1.

$$\text{Total guesses required} = \text{Total sequences} + \text{Duplicate Guesses}$$

$$\text{Total guesses required} = 1296 + d$$

Unresolved Problems

Appendix 1 - Full Code Stack

Listing 2 stage2.txt

```
getcode:
    // store address of where the function was called from
    MOV R8, LR
    getcodeNested:
        // Read input of code
        MOV R12, #tempcode
        STR R12, .ReadString
        // Validate Secret Code
        // First Character
            // Store the address of the first byte of R12 content (secret code) in R9
            LDRB R9, [R12]
            BL validateChar
        // Second Character
            // Store the address of the second byte of R12 content (secret code) in R9
            //one character is one byte so when adding one byte to R12 it will be the addres
            LDRB R9, [R12, #1]
            BL validateChar
        // Third Character
            // Store the address of the third byte of R12 content (secret code) in R9
            LDRB R9, [R12, #2]
            BL validateChar
        // Fourth Character
            // Store the address of the fourth byte of R12 content (secret code) in R9
            LDRB R9, [R12, #3]
            BL validateChar
        // Fifth Character
            // Store the address of the fifth byte of R12 content (secret code) in R9
            LDRB R9, [R12, #4]
            CMP R9, #0      //check if a character was not entered
            BNE overLimit   //if a character was entered branch to 'overLimit'
        //if a fifth character was not entered and all prior checks passed, input is valid,
        // return address the function was called from to LR
        MOV LR, R8
        B Return

invalidChar:
    MOV R10, #errorMsg1
    STR R10, .WriteString
    b getcodeNested
tooFewChar:
    MOV R10, #errorMsg2
    STR R10, .WriteString
    b getcodeNested
overLimit:
    MOV R10, #errorMsg3
    STR R10, .WriteString
    b getcodeNested

// VALIDATE CHARACTER FUNCTION
validateChar:
    CMP R9, #0        //check if a character was not entered
```

Listing 3 `codeToArray function of 'stage3.txt'`

```
// Store code to array function
// R12 - Address to tempcode is stored here
// R9 - Current Character
// R6 - Memory address of the array to fill
// R7 - Array index
secretCodeToArray:
    // load the address of the secret code into R6
    MOV R6, #secretcode
    B codeToArray
codeToArray:
    // initialize the array position to 0
    MOV R7, #0
    fillArrayLoop:
        // divide R7 (index) by 4
        LSR R7, R7, #2
        // load character into R9
        LDRB R9, [R12 + R7]
        // multiply R7 (index) by 4
        LSL R7, R7, #2

        // store character into array element
        STR R9, [R6 + R7]

        // increment index counter by 4
        ADD R7, R7, #4

        CMP R7, #codeArraySize // repeat until 4 elements of the array have been filled
        BLT fillArrayLoop
    B Return
```

Listing 4 exert from updated getcode function in 'stage3.txt'

```
getcodeNested:
        // Read input of code
        MOV R12, #tempcode
        // Initialize R6
        MOV R6, #0
        MOV R6, #secretcode
        MOV R9, #0
        LDRB R9, [R6]
        CMP R9, #0
        BEQ secretcodeentry
        BNE querycodeentry
        // If codemaker's turn
        secretcodeentry:
            STR R12, .ReadSecret
            B validateCharLoop
        // If codebreaker's turn
        querycodeentry:
            STR R12, .ReadString
            B validateCharLoop
```

Listing 5 query loop function

```
queryloop:
    // Initialize to currentGuessCount
    MOV R3, #0
    LDRB R3, currentGuessCount
    // Increment guess count by 1
    ADD R3, R3, #1
    STRB R3, currentGuessCount
    // Check if we are at guess limit
    CMP R3, R11
    BGT break
    // reset R3
    MOV R3, #0
    //
    // Continue to guess now that we've checked guess count
    // Print 'What is your guess'
    MOV R10, #requestGuessMsg
    STR R10, .WriteString
    // Print codebreaker name
    MOV R10, #codeBreaker
    STR R10, .WriteString
    // Print question mark
    MOV R10, #questionMarkMsg
    STR R10, .WriteString
    // End line
    MOV R10, #newLineMsg
    STR R10, .WriteString
    //
    // Print 'This is guess number: '
    MOV R10, #guessNumberCountMsg
    STR R10, .WriteString
    // Print guess number
    LDRB R10, currentGuessCount
    STR R10, .WriteUnsignedNum
    // End line
    MOV R10, #newLineMsg
    STR R10, .WriteString
    //
    // Get codebreaker's guess
    BL getcode
    BL queryCodeToArray

    B query
// out of guesses
break:
    HALT
//============================================================================================
// Continue stage 5 here
    query:

        B queryloop
```

Listing 6 compare codes function

```
comparecodes:
    // Initializing registers
    MOV R0, #0  // Case 1 Counter
    MOV R1, #0  // Case 2 Counter
    MOV R3, #0
    LDRB R3, arraySize // Array Size
    MOV R9, #0  // Query character
    MOV R4, #0  // Secret character
    MOV R5, #querycode  // Query array address
    MOV R6, #secretcode  // Secret array address
    MOV R7, #0  // array index / loop counter
    // R2 - Inner index

    // Case 1
    case1start:
        // initialize R2 (inner index)
        MOV R2, #0
        // Load a char from query code into R9
        LDRB R9, [R5 + R7]
        //
        // Load a char from secret code into R4
        LDRB R4, [R6 + R7]
        //
        // Compare for Case 1 (BEQ)
        CMP R4, R9
        // If case 1 is true
        BEQ case1true
        // If case 1 is false
        B case2start

    // Case 2
    case2start:
        // if main index = inner index, skip case2 check
        CMP R2, R7
        BEQ case2loopback
        // load secret char
        LDRB R4, [R6 + R2]
        // Compare secret char to query char
        CMP R4, R9
        // if case 2 is true
        BEQ case2true
        //
        // branch here to skip comparison of chars already done in case 1
        case2loopback:
        // increment inner index
        ADD R2, R2, #4

        // loop until full array checked
        CMP R2, R3
        BLT case2start
        B charQueryEnd
```

Listing 7 guess feedback function

```
guessfeedback:
    // Display position matches message
    MOV R10, #positionMatchesMsg
    STR R10, .WriteString
    // Display case 1 counter
    STR R0, .WriteUnsignedNum
    // Display colour matches message
    MOV R10, #colourMatchesMsg
    STR R10, .WriteString
    // Display case 2 counter
    STR R1, .WriteUnsignedNum
    // If 4 position matches
    CMP R0, #4
    // Branch to winstate
    BEQ winstate
    //
    // Initialize to currentGuessCount
    MOV R3, #0
    LDRB R3, currentGuessCount
    // If this was final guess
    CMP R3, R11
    BEQ losestate
    // Else, loop back for another guess
    B queryloop

    winstate:
        MOV R10, #newLineMsg
        STR R10, .WriteString
        MOV R10, #winStateMsg1
        STR R10, .WriteString
        MOV R10, #codeBreaker
        STR R10, .WriteString
        MOV R10, #winStateMsg2
        STR R10, .WriteString
        B gameover

    losestate:
        MOV R10, #newLineMsg
        STR R10, .WriteString
        MOV R10, #loseStateMsg1
        STR R10, .WriteString
        MOV R10, #codeBreaker
        STR R10, .WriteString
        MOV R10, #loseStateMsg2
        STR R10, .WriteString
        B gameover

    gameover:
        // clean our words if we wish to re-run
        BL clean
        MOV R10, #newLineMsg
```

Listing 8 `mastermind.asm`

```
//===========================================================================================
// COS10031 - Computer Technology | Assessment 3
// Vandy Aum, Marcus Mifsud, Nicole Reichert, Luke Byrnes
//  ___  ___          _                      _           _
//  |  \/  |         | |                    (_)         | |
//  | .  . | __ _ ___| |_ ___ _ __ _ __ ___  _ _ __   __| |
//  | |\/| |/ _` / __| __/ _ \ '__| '_ ` _ \| | '_ \ / _` |
//  | |  | | (_| \__ \ ||  __/ |  | | | | | | | | | | (_| |
//  \_|  |_/\__,_|___/\__\___|_|  |_| |_| |_|_|_| |_|\__,_|
//
// Register Assignations
    // R0 (Compare Code of Correct Pos/Col)
    // R1 (Compare Code of (Correct Pos, Incorrect Col))
    // R2
    // R3
    // R4
    // R5
    // R6
    // R7
    // R8 Function Return (stores LR to return after a function is used within a function)
    // R9 Code character address
    // R10 String Handling
    // R11 Guess Limit
    // R12 Address to temp code
//===========================================================================================
// Stage 1 - Game Setup - Luke Byrnes & Nicole Reichert
//
// Prompt and store Codemaker Name
    // Set whoIsCodeMakerMsg Query prompt to R10
    MOV R10, #whoIsCodeMakerMsg
    // print whoIsCodeMakerMsg Query from R10
    STR R10, .WriteString
    // Move codeMaker address to R10
    MOV R10, #codeMaker
    // Take input from user and store to R10
    STR R10, .ReadString
    // Print codeMaker value from R10
    STR R10, .WriteString
//
// Prompt and store CodeBreaker Name
    // Set whoIsCodeMakerMsg Query prompt to R10
    MOV R10, #whoIsCodeBreakerMsg
    // print whoIsCodeMakerMsg Query from R10
    STR R10, .WriteString
    // Move codeBreaker address to R10
    MOV R10, #codeBreaker
    // Take input from user and store to R10
    STR R10, .ReadString
    // Print codeMaker value from R10
    STR R10, .WriteString
//
```