



COS10004 – COMPUTER SYSTEM

Lab Session: Monday 17:30 – 19:30 | Assignment 2 – Mastermind
Game

ABSTRACT

Levish Boodhoo

Student ID: 103818390

Mastermind – Stage 1

Game Setup

The screenshot shows the ARMLite Simulator V1.2a interface. The Program window contains the following assembly code:

```
1 //R0 used for writing messages
2 //R2 is the dedicated register to store the number of allowed guesses
3 MOV R0, #msg1
4 STR R0, #codebreaker //Stores codebreaker's name
5 MOV R0, #codebreaker //Stores codebreaker's name
6 STR R0, #codebreaker
7 MOV R1, #codebreaker
8 STR R1, #codebreaker
9 MOV R0, #0x0A //New line character code
10 STRB R0, #codebreaker
11 MOV R0, #msg2
12 STR R0, #codemaker //Stores codemaker's name
13 MOV R0, #codemaker
14 STR R0, #codemaker
15 MOV R1, #codemaker
16 STR R1, #codemaker
17 MOV R0, #0x0A //New line character code
18 STRB R0, #codemaker
19 MOV R0, #msg3
20 STR R0, #codemaker
21 LDR R2, #inputnum
22 STR R2, #storeamount //store amount of guesses
23 HALT
24 msg1: .ASCII "Codebreaker is "
25 codebreaker: .BLOCK 128
26 msg2: .ASCII "Codemaker is "
27 codemaker: .BLOCK 128
28 msg3: .ASCII "Maximum number of guesses: "
29 readnum: .BLOCK 128
```

The Processor window shows the following registers:

Register	Value
PC	0x00000000
LR	0x00000000
SP	0x00100000
R12	0x00000000
R11	0x00000000
R10	0x00000000
R9	0x00000000
R8	0x00000000
R7	0x00000000
R6	0x00000000
R5	0x00000000
R4	0x00000000
R3	0x00000000
R2	0x00000000
R1	0x00000000
R0	0x00000000

The Memory window shows a memory dump starting at 0x00000000. The Input/Output window shows the program assembled and ready to run.

The three screenshots show the game flow:

- Initial Setup:** The codebreaker's name is "Levish" and the codemaker's name is "Neil". The maximum number of guesses is 10.
- Input for Codebreaker's Name:** The user enters "Levish" as the codebreaker's name.
- Output of the Game:** The output shows "Codebreaker is Levish" and "Codemaker is Neil".

Flow of code asking for input and displaying outputs.

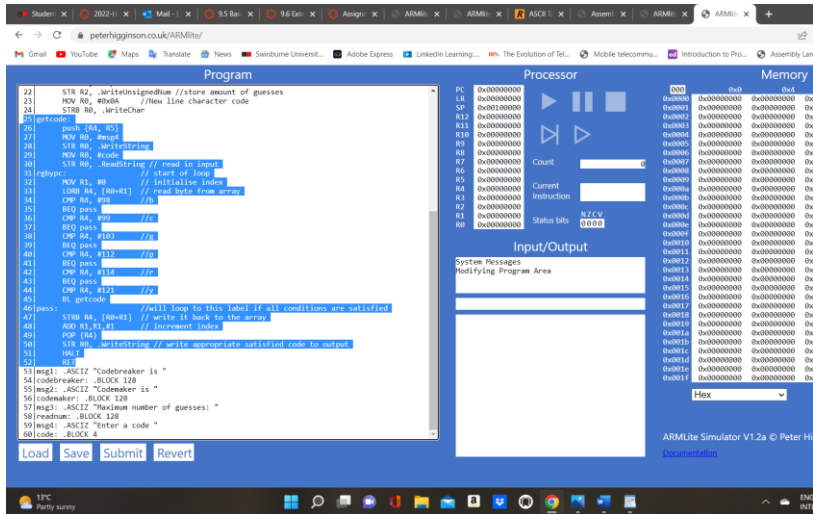
Input asking for the number of queries for future use.

Output lines asking for information to begin the game

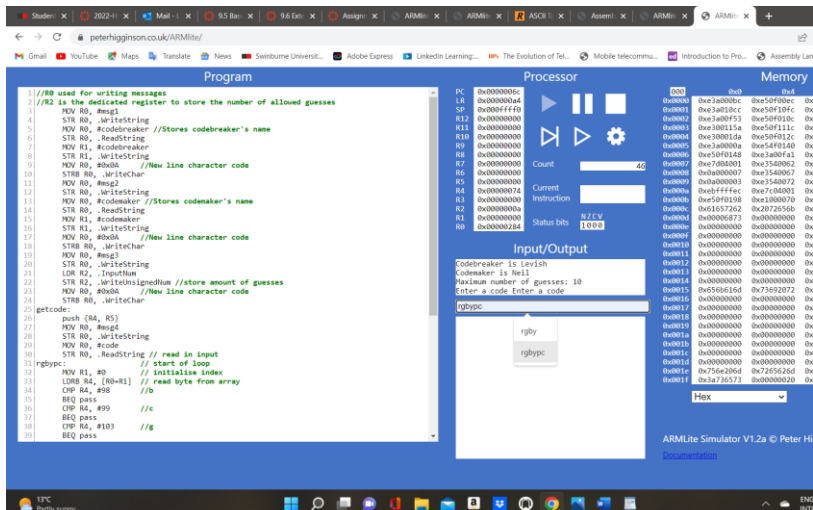
In this context, R0 is used to write messages or commands on the output display which asks for names and attempt numbers. However, there are labels used to store names entered by players such as #codebreaker and #codemaker. As far as query numbers are concerned, I have used a dedicated register to store the number (R2) which will be used only for the query number throughout the whole assembly code. As the code is processed stepwise, it will ask for inputs and as we press Enter, the line "Codebreaker is <name>" and other following lines will be displayed.

Mastermind – Stage 2

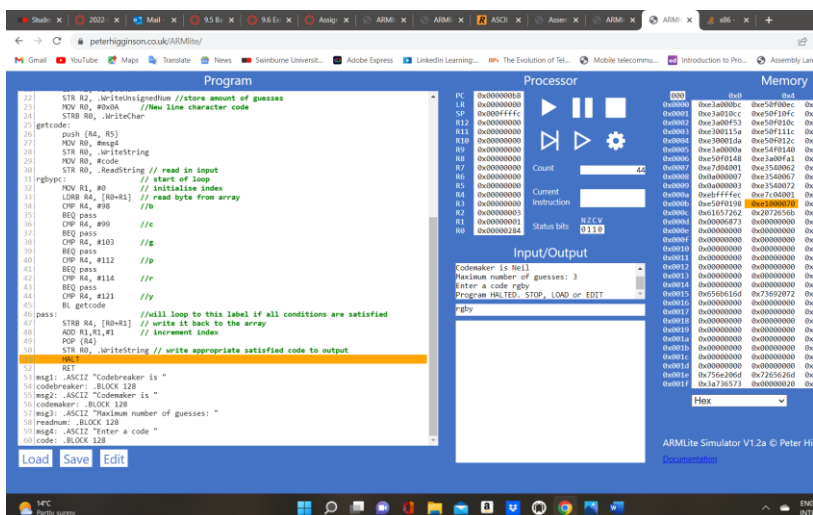
A code entry Function



This is the code for Stage2. There has been a functionality that I have faced problems creating which is limiting the code count to 4. However, I was able to define all the required code to allow the player to input code that consists only of "rgbypc".



In this figure, I have tried to input alphabet other than "rgbypc" and as soon as the code get a code which does not satisfies the mentioned conditions, it will loop back to the start of the function, "getcode". There are other labels like "pass and" rgbypc"



This is the whole code functioning well. Only the 4-length code has been a limitation to me. Otherwise, the whole other code satisfies the required concept. It reads in the required lowercase alphabet code to display it after the line.

Mastermind – Stage 3

Getting the Secret Code

The screenshot shows the ARM simulator interface. On the left, the assembly code is displayed, with lines 25-34 highlighted. The code includes instructions for writing a message, reading a secret code, and a loop to process the code. On the right, the 'Input/Output' window shows the text 'Maximum number of guesses: 4' and 'Neil, please enter a 4-character secret code'. The 'Enter a code' field contains the text 'rgby'.

Program

```
25 MOV R1, #codemaker
26 STR R1, .WriteString //write codemaker's name
27 MOV R0, #msg5 // display message asking for secretcode
28 STR R0, .WriteString
29 MOV R0, #secretcode
30 MOV R0, #0x0A //New line character code
31 STRB R0, .WriteChar
32 BLI getcode //loop to label in getcode function to process secret code
33 STR R0, .ReadString // read in input
34 HALT
35 getcode:
36 MOV R0, #msg4
37 STR R0, .WriteString
38 MOV R0, #code
39 STR R0, .ReadString // read in input
40 rgbyc: // start of loop
41 PUSH {R4}
42 MOV R1, #0 // initialise index
43 LDRB R4, [R0+R1] // read byte from array
44 CMP R4, #98 //b
45 BEQ pass
46 CMP R4, #99 //c
47 BEQ pass
48 CMP R4, #103 //p
49 BEQ pass
50 CMP R4, #112 //p
51 BEQ pass
52 CMP R4, #114 //r
53 BEQ pass
54 CMP R4, #121 //y
55 BNE getcode
56 pass:
57 STRB R4, [R0+R1] //will loop to this label if all conditions are satisfied
58 POP {R4}
59 ADD R1, R1, #1 // increment index
60 STR R0, .WriteString // write appropriate satisfied code to output
61 HALT
62 RET
63 msg1: .ASCII "Codebreaker is "
64 codebreaker: .BLOCK 128
65 msg2: .ASCII "Codemaker is "
66 codemaker: .BLOCK 128
67 msg3: .ASCII "Maximum number of guesses: "
68 readnum: .BLOCK 128
69 msg4: .ASCII "Enter a code "
70 code: .BLOCK 4
71 msg5: .ASCII " , please enter a 4-character secret code "
72 secretcode: .BLOCK 4
```

Current Instruction: NZCV 0110

Status bits: NZCV 0110

Input/Output

Maximum number of guesses: 4

Neil, please enter a 4-character secret code

Enter a code

rgby

Hex

The displayed code asks the codemaker for the secret code and storing the information in an array called, #secretcode.

It loops back to the function, "getcode" to allow input of secret code and then, processing the conditions applied to the secret code.

The screenshot shows the ARM simulator interface. On the left, the assembly code is displayed, with lines 34-62 highlighted. The code includes instructions for writing a message, reading a secret code, and a loop to process the code. On the right, the 'Input/Output' window shows the text 'Maximum number of guesses: 4' and 'Neil, please enter a 4-character secret code'. The 'Enter a code' field contains the text 'rgby'.

Program

```
34 HALT
35 getcode:
36 MOV R0, #msg4
37 STR R0, .WriteString
38 MOV R0, #code
39 STR R0, .ReadString // read in input
40 rgbyc: // start of loop
41 PUSH {R4}
42 MOV R1, #0 // initialise index
43 LDRB R4, [R0+R1] // read byte from array
44 CMP R4, #98 //b
45 BEQ pass
46 CMP R4, #99 //c
47 BEQ pass
48 CMP R4, #103 //p
49 BEQ pass
50 CMP R4, #112 //p
51 BEQ pass
52 CMP R4, #114 //r
53 BEQ pass
54 CMP R4, #121 //y
55 BNE getcode
56 pass:
57 STRB R4, [R0+R1] //will loop to this label if all conditions are satisfied
58 POP {R4}
59 ADD R1, R1, #1 // increment index
60 STR R0, .WriteString // write appropriate satisfied code to output
61 HALT
62 RET
63 msg1: .ASCII "Codebreaker is "
64 codebreaker: .BLOCK 128
65 msg2: .ASCII "Codemaker is "
66 codemaker: .BLOCK 128
67 msg3: .ASCII "Maximum number of guesses: "
68 readnum: .BLOCK 128
69 msg4: .ASCII "Enter a code "
70 code: .BLOCK 4
71 msg5: .ASCII " , please enter a 4-character secret code "
72 secretcode: .BLOCK 4
```

Processor

Count: 52

Current Instruction: NZCV 0110

Status bits: NZCV 0110

Input/Output

Neil, please enter a 4-character secret code

Enter a code rgby

Program HALTED. STOP, LOAD or EDIT

rgby

Hex

Clear

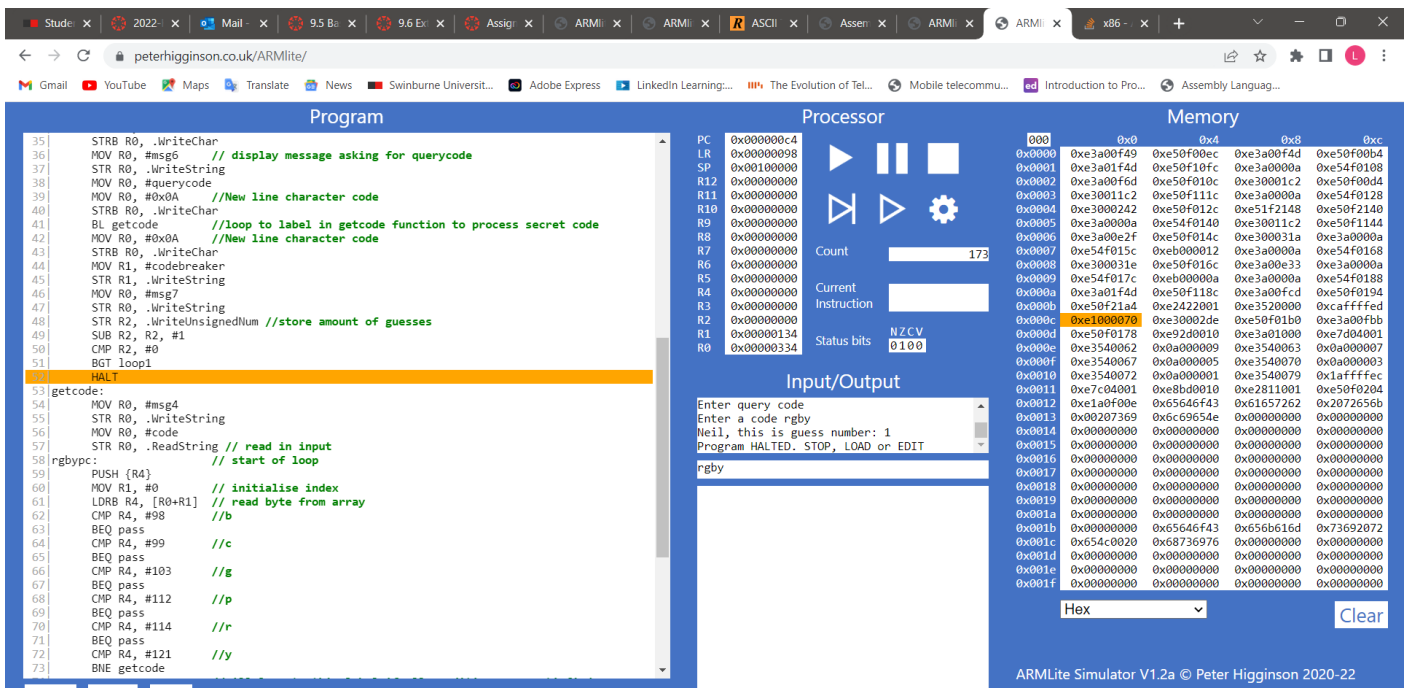
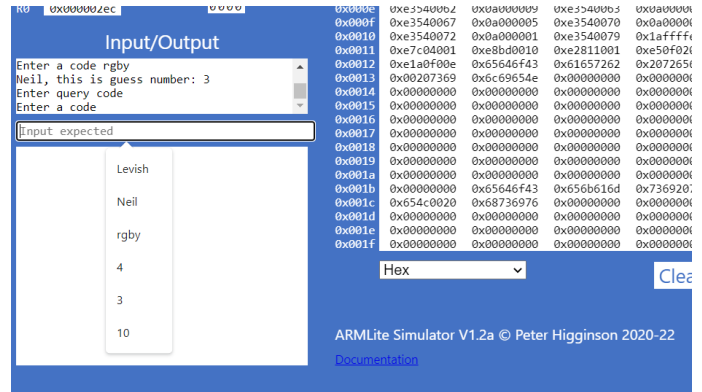
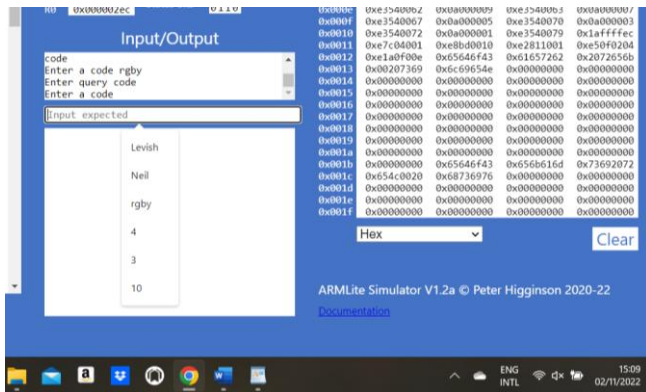
ARMLite Simulator V1.2a © Peter Higginson 2020-22

Documentation

As the code is processed, after asking for the information to begin the name, it will ask for the codemaker to enter his/her secret code. Then, after displaying the command on the output box, the player will have to enter a secret code of the alphabet "rgbyc". After writing the code, the code will loop to the "getcode" function to process the structure of the code to see whether it satisfies the required and mentioned conditions. If it does not satisfy the conditions, it will keep on looping to the "getcode" function otherwise, it will keep on running till the code is halted.

Mastermind – Stage 4

Query Code Entry



Just after the codemaker has entered his/ her secretcode, the game is on and the codebreaker has to figure out the same exact code that has been entered by his/her opponent. There the codebreaker get his/her chance to play next by getting to input a code followed after a command telling him/her to enter a query code. Now, to monitor the number of chances that the codebreaker has, we will use that dedicated register to keep account of the chances that are left. As the codebreaker enters a query code, he /she will get an alert, telling him/her how many chances are left and once the number of queries has reached its threshold, the program will halt.

Mastermind – Stage 5a

Query Code evaluation

The screenshot displays the ARMLite Simulator V1.2a interface. The main window shows the assembly code for the program, which is a Mastermind game logic. The code includes instructions for pushing and popping registers, comparing values, and branching based on the result. The Processor window shows the current instruction is 'BGT loop1' and the Count register is 4. The Memory window shows the secret code at address 0x3a00f55 is 0x3a00f59, 0x3a00f59, 0x3a00f59, 0x3a00f59. The Input/Output window shows 'Input expected'.

decode that secret colored code until he/she wins or loses and this decision totally depends on the number of queries the codebreaker has been given. So, to determine the success or failure of the codebreaker, the secretcode and the querycode need to be compared and verified whether the color count and the position of the colored pegs are the same as the secret code. However, to do so, a function, (comaprecodes) has been added to compare all the mentioned attributes of the secretcode. There are various scenarios and this has been subdivided into case 1 and case 2 in the assignment itself. Therefore, case 1 will verify the number of colors which match, and it will be stored in R0. Case 2 determines and counts the number of pegs that are in the correct position and the count is stored in R1.

Query Code evaluation

Gmail
 Maps
 Translate
 News
 Winburne University...
 Adobe Express
 LinkedIn Learning...
 The Evolution of Tel...
 Mobile Telecom...
 Introduction to Pro...
 Assembly Language...

Program

```

63:equal:
64:  ADD R3,R3,#1
65:  MOV R0,R3
66:  CMP R0,R4
67:  MOV R3,#msg8
68:  STR R3, .WriteString
69:  STR R0, .WriteUnsignedNum
70:  MOV R3,#msg9
71:  STR R3, .WriteString
72:  STR R0, .WriteUnsignedNum
73:  CMP R1,R4
74:  BEQ win
75:  POP {R4}
76:  SUB R2,R2,#1
77:  CMP R2,#0
78:  BGT loop1
79:  MOV R0,#0x0A //New line character code
80:  STRB R0, .WriteChar
81:  MOV R1,#codebreaker
82:  STR R1, .WriteString
83:  MOV R3,#msg10
84:  STR R3, .WriteString
85:  MOV R0,#0x0A //New line character code
86:  STRB R0, .WriteChar
87:  MOV R3,#msg12
88:  STR R3, .WriteString
89:  HALT
90:win:
91:  MOV R0,#0x0A //New line character code
92:  STRB R0, .WriteChar
93:  MOV R1,#codebreaker
94:  STR R1, .WriteString
95:  MOV R3,#msg11
96:  STR R3, .WriteString
97:  MOV R0,#0x0A //New line character code
98:  STRB R0, .WriteChar
99:  MOV R3,#msg12
100:  STR R3, .WriteString
101:  HALT
102:retcode:

```

Load

Save

Edit

Processor

PC

0x0000014C

LR

0x00000098

SP

0x00100000

R12

0x00000000

R11

0x00000000

R10

0x00000000

R9

0x00000000

R8

0x00000000

R7

0x00000000

R6

0x00000000

R5

0x00000000

R4

0x00000000

R3

0x00000043e

R2

0x00000000

R1

0x0000001e8

R0

0x00000000a

Count

185

Current Instruction

NZCV

0100

Status bits

NZCV

0100

Memory

000	0x0	0x4	0x8	0xc
0xe0000	0xe3a00f76	0xe50f00ec	0xe3a00f7a	0xe50f00b4
0xe0001	0xe3a01f7a	0xe50f10fc	0xe3a0000a	0xe54f0108
0xe0002	0xe3a00f9a	0xe50f010c	0xe3000276	0xe50f00d4
0xe0003	0xe3001276	0xe50f111c	0xe3a0000a	0xe54f0128
0xe0004	0xe30002f6	0xe50f012c	0xe51f2148	0xe50f2140
0xe0005	0xe3a0020a	0xe54f0140	0xe50f1144	0xe50f1144
0xe0006	0xe3a00fe9	0xe50f014c	0xe30003ce	0xe3a0000a
0xe0007	0xe54f015c	0xeb00003f	0xe3a0000a	0xe54f0168
0xe0008	0xe30003d2	0xe50f016c	0xe3a00ff9	0xe3a0000a
0xe0009	0xe54f017c	0xeb000017	0xe3a00000	0xe54f0188
0xe000a	0xe3a01f7a	0xe50f118c	0xe3a00ff9	0xe50f1034
0xe000b	0xe50f214a	0xe30003ce	0xe3a01ff9	0xe3a03000
0xe000c	0xe92d0010	0xe8000100	0xe7d14003	0xe1540000
0xe000d	0x0a000000	0xe2833001	0xea1a1003	0xe1510004
0xe000e	0xea0fffff	0xe2833001	0xea1a0003	0xe1500004
0xe000f	0xe3003401	0xe50f21dc	0xe50f01ec	0xe3003414
0xe0010	0xe50f31e8	0xe50f11f8	0xe251000a	0xe300000e
0xe0011	0xe8bd0010	0xe4220001	0xe5250000	0xcacff455
0xe0012	0xe3a0000a	0xe54f0210	0xe3a01f7a	0xe50f1214
0xe0013	0xe3003427	0xe50f321c	0xe3a0000a	0xe54f0228
0xe0014	0xe300342e	0xe50f322c	0xe1000070	0xe3a0000a
0xe0015	0xe54f023c	0xe3a01f7a	0xe3003433	0xe3003433
0xe0016	0xe50f3248	0xe3a0000a	0xe54f022a	0xe300343e
0xe0017	0xe50f3258	0xe1000070	0xe3000392	0xe50f0264
0xe0018	0xe3a00e3a	0xe50f022c	0xe92d0010	0xe3a01000
0xe0019	0x7d04001	0xe3540062	0x0a000009	0xe3540063
0xe001a	0x0a000007	0xe3540067	0x0a000005	0xe3540078
0xe001b	0x0a000003	0xe3540072	0x0a000001	0xe3540079
0xe001c	0x1affffcc	0xe7d04001	0xe8bd0010	0xe2510100
0xe001d	0xe50f02b8	0xe656ff43	0xe566ff43	0xe5672662
0xe001e	0x207265b0	0x00207369	0x6976654c	0x00000673
0xe001f	0x00000000	0x00000000	0x00000000	0x00000000

Input/Output

matches: 2 , Colour matches: 1

Levish, you LOSE!

Game Over!

Program HALTED. STOP, LOAD or EDIT

bgry

Hex

Clear

ARMLite Simulator V1.2a © Peter Higgins 2020-22

[Documentation](#)

Program

```

108:    PUSH {R4}
109:    MOV R1, #0           // initialise index
110:    LDRB R4, [R0+R1]     // read byte from array
111:    CMP R4, #98         //b
112:    BEQ pass
113:    CMP R4, #99         //c
114:    BEQ pass
115:    CMP R4, #103        //g
116:    BEQ pass
117:    CMP R4, #112        //p
118:    BEQ pass
119:    CMP R4, #114        //r
120:    BEQ pass
121:    CMP R4, #121        //y
122:    BNE getcode
123:pass:                    //will loop to this label if all conditions are satisfied
124:    STRB R4, [R0+R1]     // write it back to the array
125:    POP {R4}
126:    ADD R1,R1,#1         // increment index
127:    STR R0, .WriteString // write appropriate satisfied code to output
128:    RET
129:
130:msg1: .ASCIIZ "Codebreaker is "
131:codebreaker: .BLOCK 128
132:msg2: .ASCIIZ "Codemaker is "
133:codemaker: .BLOCK 128
134:msg3: .ASCIIZ "Maximum number of guesses: "
135:readnum: .BLOCK 128
136:msg4: .ASCIIZ "Enter a code "
137:code: .BLOCK 4
138:msg5: .ASCIIZ ", please enter a 4-character secret code "
139:secretcode: .BLOCK 4
140:msg6: .ASCIIZ "Enter query code "
141:querycode: .BLOCK 4
142:msg7: .ASCIIZ ", this is guess number: "
143:msg8: .ASCIIZ "Position matches: "
144:msg9: .ASCIIZ ", Colour matches: "
145:msg10: .ASCIIZ ", you LOSE!"
146:msg11: .ASCIIZ ", you WIN!"
147:msg12: .ASCIIZ "Game Over!"

```

Processor

PC

0x0000014c

LR

0x00000098

SP

0x00100000

R12

0x00000000

R11

0x00000000

R10

0x00000000

R9

0x00000000

R8

0x00000000

R7

0x00000000

R6

0x00000000

R5

0x00000000

R4

0x00000000

R3

0x0000043e

R2

0x00000000

R1

0x000001e8

R0

0x0000000a

Count

185

Current Instruction

NZCV 0100

Status bits

NZCV 0100

Memory

000	0x0	0x4	0x8	0xc
0x0000	0xe3a00f76	0xe5f0f0ec	0xe3a00f7a	0xe5f0f004
0x0001	0xe3a01f7a	0xe5f0f10c	0xe3a0000a	0xe5f4f108
0x0002	0xe3a00f9a	0xe5f0f10c	0xe3000276	0xe5f0f004
0x0003	0xe3001276	0xe5f0f11c	0xe3a0000a	0xe5f4f128
0x0004	0xe30002f6	0xe5f0f12c	0xe5f12148	0xe5f0f210
0x0005	0xe3a0000a	0xe5f4f010	0xe3001276	0xe5f0f144
0x0006	0xe3a00fe9	0xe5f0f01c	0xe30003ce	0xe3a0000a
0x0007	0xe5f4f015c	0xeb00003f	0xe3a0000a	0xe5f4f018
0x0008	0xe30003d3	0xe5f0f015c	0xe3a0000a	0xe3000000
0x0009	0xe5f4f015c	0xeb00003f	0xe3a0000a	0xe5f4f108
0x000a	0xe3a01f7a	0xe5f0f118	0xe3a00ffa	0xe5f0f194
0x000b	0xe5f0f21a4	0xe30003ce	0xe3a01ff9	0xe3a03000
0x000c	0xe92d0010	0xe0001001	0xe7d14003	0xe5150000
0x000d	0x0a000003	0xe2833001	0x1ea10103	0xe5150004
0x000e	0xe4fffff1	0xe2833001	0x1ea00003	0xe5150004
0x000f	0xe3003401	0xe5f0f31dc	0xe5f0f1ec	0xe3003414
0x0010	0xe5f0f1e8	0xe3515000	0xe3000000	0xe3000000
0x0011	0xe8bd0010	0xe2422001	0xe5320000	0xe4fffff5
0x0012	0xe3a0000a	0xe5f4f010	0xe3a01f7a	0xe5f0f124
0x0013	0xe3003427	0xe5f0f321c	0xe3a0000a	0xe5f4f028
0x0014	0xe300342c	0xe5f0f322c	0xe3a0000a	0xe3a0000a
0x0015	0xe5f4f023c	0xe3a01f7a	0xe5f0f120	0xe3003433
0x0016	0xe5f0f3248	0xe3a0000a	0xe5f4f024	0xe300343e
0x0017	0xe5f0f3258	0xe1000070	0xe3000392	0xe5f0f024
0x0018	0xe3a0003a	0xe5f0f022c	0xe3a01200	0xe3a01000
0x0019	0xe7d04001	0xe3540062	0xe0a00009	0xe3540063
0x001a	0x0a000007	0xe3540067	0x0a000005	0xe3540070
0x001b	0x0a000003	0xe3540072	0x0a000001	0xe3540079
0x001c	0x1afffff8	0xe7d04001	0xe8bd0010	0xe2811001
0x001d	0xe5f0f02b8	0xe1a0f00e	0x6564f013	0x61657262
0x001e	0x2072656b	0x00207369	0x6976654c	0x00006873
0x001f	0x00000000	0x00000000	0x00000000	0x00000000

Input/Output

matches: 2 , Colour matches: 1

Levish, you LOSE!

Game Over!

Program HALTED. STOP, LOAD or EDIT

bgr

Hex

Clear

Load

Save

Edit

ARMLite Simulator V1.2a © Peter Higginson 2020-22

Documentation

After implementing that function where it compares the query code and the secret code to count the number of pegs of the same color and that is positioned in the right order according to the attributes of the secret code, other lines are required to complete the code and the game. That is, the numbers have been stored in R0 and R1. Therefore, for the codebreaker to have an overview of his/her move about the order and color of pegs, the information stored in R0 and R1 need to be displayed and this must loop back to the querycode input every time the querycode does not match the secret code and thus determine whether the codebreaker win the game or lose. To do this, several displayed lines have been implemented to ensure a fair end of the game. This is Mastermind Game.