

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

Contents

Mastermind Assembly Game	3
Program Overview	3
Key Functions	3
Stage 1 (stage1.txt)	3
Stage 2 (stage2.txt)	4
Stage 3 (stage3.txt)	7
Stage 4 (stage4.txt)	8
Stage 5a (stage5a.txt)	11
Stage 5b (stage5b.txt)	14
Assumptions	17
No restrictions for user submitted Guess Limit	17
No Duplicate Guess controls	17
Appendix - Full Code Stack	18

Mastermind Assembly Game

Program Overview

This program replicates gameplay of the Mastermind boardgame in Assembly using the ARMLite assembly utility.

Key Functions

Stage 1 (stage1.txt)

Stage 1 request's the players to enter their names and define the maximum number of guesses using the following functions.

File: stage1.txt

```
56: // Display whoIsCodeMaker Query prompt:
57: whoIsCodeMakerMsg: .ASCIIZ "Codemaker is: "
58: // Store block of memory of 128 bytes to store the string
59: codeMaker: .BLOCK 128
60: // Display whoIsCodeMaker Query prompt:
61: whoIsCodeBreakerMsg: .ASCIIZ "\nCodebreaker is: "
62: // Store block of memory of 128 bytes to store the string
63: codeBreaker: .BLOCK 128
64: // Display guessLimit Query prompt:
65: whatIsGuessLimitMsg: .ASCIIZ "\nGuess Limit: "
66: // Store the guesslimit as a label, 0 as default
67: guessLimit: .WORD 0
```

The screenshot displays the ARMLite Simulator V1.3.1 interface, which is divided into three main sections: Program, Processor, and Memory.

Program Section: This section shows the assembly code for the program. The code is as follows:

```
22 // Set codeMaker value to R4
23 MOV R4, #codeMaker
24 // Take input from user and store to R4
25 STR R4, .ReadString
26 // Print codeMaker value from R4
27 STR R4, .WriteString
28 // Prompt and store CodeBreaker Name
29 MOV R10, #whoIsCodeBreaker
30 // print whoIsCodeMaker Query from R4
31 STR R10, .WriteString
32 // Set codeMaker value to R5
33 MOV R5, #codeBreaker
34 // Take input from user and store to R5
35 STR R5, .ReadString
36 // Print codeMaker value from R5
37 STR R5, .WriteString
38 // Prompt and store GuessLimit for the session
39 // Set guessLimit Query prompt to R10
40 MOV R10, #whatIsGuessLimit
41 // Print whatIsGuessLimit from R10
42 STR R10, .WriteString
43 // Take input from user and store to R11
44 LDR R11, .InputNum
45 // Print guessLimit from R11
46 STR R11, .WriteSignedNum
47 // Stop Program.
48 HALT
49 // Program functions:
50 // Display whoIsCodeMaker Query prompt:
51 whoIsCodeMaker: .ASCIIZ "Codemaker is: "
52 // Store block of memory of 128 bytes to store the string
53 codeMaker: .BLOCK 128
54 // Display whoIsCodeMaker Query prompt:
55 whoIsCodeBreaker: .ASCIIZ "\nCodebreaker is: "
56 // Store block of memory of 128 bytes to store the string
57 codeBreaker: .BLOCK 128
58 // Display guessLimit Query prompt:
59 whatIsGuessLimit: .ASCIIZ "\nGuess Limit: "
```

Processor Section: This section shows the current state of the processor. The PC (Program Counter) is 0x0000003c. The LR (Link Register) is 0x00000000. The SP (Stack Pointer) is 0x00100000. The R12 register is 0x00000000. The R11 register is 0x0000000c. The R10 register is 0x00000015d. The R9 register is 0x00000000. The R8 register is 0x00000000. The R7 register is 0x00000000. The R6 register is 0x00000000. The R5 register is 0x000000dd. The R4 register is 0x0000004b. The R3 register is 0x0000000b. The R2 register is 0x00000000. The R1 register is 0x00000000. The R0 register is 0x00000000. The Count is 15. The Current Instruction is NZCV. The Status bits are 0000.

Memory Section: This section shows the memory dump. The memory is organized into rows of 16 bytes each. The first row shows the memory address 0x00000000 and the value 0x00000000. The second row shows the memory address 0x00000004 and the value 0x00000000. The third row shows the memory address 0x00000008 and the value 0x00000000. The fourth row shows the memory address 0x0000000c and the value 0x00000000. The fifth row shows the memory address 0x00000010 and the value 0x00000000. The sixth row shows the memory address 0x00000014 and the value 0x00000000. The seventh row shows the memory address 0x00000018 and the value 0x00000000. The eighth row shows the memory address 0x0000001c and the value 0x00000000. The ninth row shows the memory address 0x00000020 and the value 0x00000000. The tenth row shows the memory address 0x00000024 and the value 0x00000000. The eleventh row shows the memory address 0x00000028 and the value 0x00000000. The twelfth row shows the memory address 0x0000002c and the value 0x00000000. The thirteenth row shows the memory address 0x00000030 and the value 0x00000000. The fourteenth row shows the memory address 0x00000034 and the value 0x00000000. The fifteenth row shows the memory address 0x00000038 and the value 0x00000000. The sixteenth row shows the memory address 0x0000003c and the value 0x00000000.

Input/Output Section: This section shows the input and output of the program. The input is "Luke" and the output is "Vandy". The program is currently in the "HALTED" state.

Buttons: The interface includes buttons for "Load", "Save", and "Edit".

Footer: The footer text reads "ARMLite Simulator V1.3.1 © Peter Higginson 2020-24 Documentation".

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.

File: stage2.txt

```
080: getcode:
081:     // store address of where the function was called from
082:     MOV R8, LR
083:     // branch here if the code entered is invalid
084:     getcodeNested:
085:         // Read input of code
086:         MOV R12, #tempcode
087:         STR R12, .ReadString
088:         // Validate Secret Code
089:         // First Character
090:         // Store the address of the first byte of R12 content
            (secret code) in R9
091:         LDRB R9, [R12]
092:         BL validateChar
093:         // Second Character
094:         // Store the address of the second byte of R12 content
            (secret code) in R9
095:         //one character is one byte so when adding one byte to R12
            it will be the address of the next character
096:         LDRB R9, [R12, #1]
097:         BL validateChar
098:         // Third Character
099:         // Store the address of the third byte of R12 content
            (secret code) in R9
100:         LDRB R9, [R12, #2]
101:         BL validateChar
102:         // Fourth Character
103:         // Store the address of the fourth byte of R12 content
            (secret code) in R9
104:         LDRB R9, [R12, #3]
105:         BL validateChar
106:         // Fifth Character
107:         // Store the address of the fifth byte of R12 content
            (secret code) in R9
108:         LDRB R9, [R12, #4]
109:         CMP R9, #0 //check if a character was not entered
110:         BNE overLimit //if a character was entered branch to
            'overLimit'
111:         //if a fifth character was not entered and all prior checks
            passed, input is valid, return to code
112:         // return address the function was called from to LR
113:         MOV LR, R8
114:         B Return
115:
```

```

116: invalidChar:
117:     MOV R10, #errorMsg1
118:     STR R10, .WriteString
119:     b getCodeNested
120: tooFewChar:
121:     MOV R10, #errorMsg2
122:     STR R10, .WriteString
123:     b getCodeNested
124: overLimit:
125:     MOV R10, #errorMsg3
126:     STR R10, .WriteString
127:     b getCodeNested
128:
129: // VALIDATE CHARACTER FUNCTION
130: validateChar:
131:     CMP R9, #0           //check if a character was not entered
132:     BEQ tooFewChar
133:     CMP R9, #0x72        //check if the character is r(red)
134:     BEQ Return
135:     CMP R9, #0x67        //check if the character is g(green)
136:     BEQ Return
137:     CMP R9, #0x62        //check if the character is b(blue)
138:     BEQ Return
139:     CMP R9, #0x79        //check if the character is y(yellow)
140:     BEQ Return
141:     CMP R9, #0x70        //check if the character is p(purple)
142:     BEQ Return
143:     CMP R9, #0x63        //check if the character is c(cyan)
144:     BEQ Return
145:     b invalidChar        //branch to 'invalidChar' if the character was not
                           //matched by any of the above checks
146:
147: // Function to return from function
148: Return: RET

```

The screenshot displays the ARMLite Simulator V1.3.1 interface, which is divided into three main panels: Program, Processor, and Memory.

- Program Panel:** Shows the assembly code being executed. The current instruction is at line 74: `MOV R10, #testMsg`. The code includes comments for various stages and functions, such as "Stage 2 - A Code Entry Function - Vandy Aum" and "FUNCTIONS".
- Processor Panel:** Displays the current state of the processor. The PC (Program Counter) is at address 0x00000058. The Count register is set to 73. The Status bits are N7CV and 0100. The Input/Output section shows the Codebreaker is "vandy", the Guess Limit is "1000", and the Enter the Code field is empty.
- Memory Panel:** Shows a hex dump of memory. The address 0x00000058 is highlighted, containing the value 0x00000058. The memory dump shows various values across different addresses, including 0x00000000, 0x00000001, 0x00000002, etc.

At the bottom of the interface, there are buttons for "Load", "Save", and "Edit". The ARMLite Simulator V1.3.1 logo and copyright information (© Peter Higginson 2020-24) are also visible.

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.

File: stage3.txt - code to array function

```
170: // Store code to array function
171: // R12 - Address to tempcode is stored here
172: // R9 - Current Character
173: // R6 - Memory address of the array to fill
174: // R7 - Array index
175: secretCodeToArray:
176:     // load the address of the secret code into R6
177:     MOV R6, #secretcode
178:     B codeToArray
179: codeToArray:
180:     // initialize the array position to 0
181:     MOV R7, #0
182:     fillArrayLoop:
183:         // divide R7 (index) by 4
184:         LSR R7, R7, #2
185:         // load character into R9
186:         LDRB R9, [R12 + R7]
187:         // multiply R7 (index) by 4
188:         LSL R7, R7, #2
189:
190:         // store character into array element
191:         STR R9, [R6 + R7]
192:
193:         // increment index counter by 4
194:         ADD R7, R7, #4
195:
196:         CMP R7, #codeArraySize // repeat until 4 elements of the array have been
filled
197:         BLT fillArrayLoop
198:     B Return
```

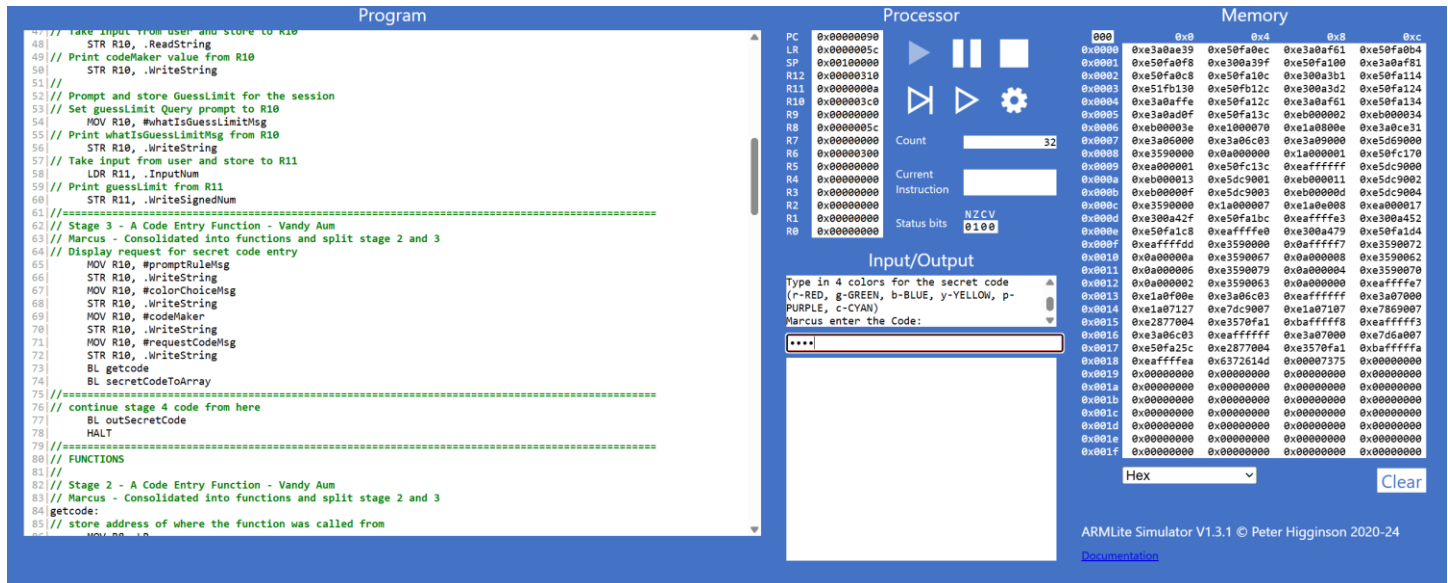
File: stage3.txt - exert from updated getcode function

```
088:     getcodeNested:
089:         // Read input of code
090:         MOV R12, #tempcode
091:         // Initialize R6
092:         MOV R6, #0
093:         MOV R6, #secretcode
094:         MOV R9, #0
095:         LDRB R9, [R6]
096:         CMP R9, #0
097:         BEQ secretcodeentry
098:         BNE querycodeentry
099:         // If codemaker's turn
100:         secretcodeentry:
```

```

101:      STR R12, .ReadSecret
102:      B validateCharLoop
103:      // If codebreaker's turn
104:      querycodeentry:
105:      STR R12, .ReadString
106:      B validateCharLoop

```



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.

File: stage4.txt - queryloop function

```

080: queryloop:
081:      // Initialize to currentGuessCount
082:      MOV R3, #0
083:      LDRB R3, currentGuessCount
084:      // Increment guess count by 1
085:      ADD R3, R3, #1
086:      STRB R3, currentGuessCount
087:      // Check if we are at guess limit
088:      CMP R3, R11
089:      BGT break
090:      // reset R3
091:      MOV R3, #0
092:      //
093:      // Continue to guess now that we've checked guess count
094:      // Print 'What is your guess'
095:      MOV R10, #requestGuessMsg
096:      STR R10, .WriteString

```



```

097:    // Print codebreaker name
098:    MOV R10, #codeBreaker
099:    STR R10, .WriteString
100:    // Print question mark
101:    MOV R10, #questionMarkMsg
102:    STR R10, .WriteString
103:    // End line
104:    MOV R10, #newLineMsg
105:    STR R10, .WriteString
106:    //
107:    // Print 'This is guess number: '
108:    MOV R10, #guessNumberCountMsg
109:    STR R10, .WriteString
110:    // Print guess number
111:    LDRB R10, currentGuessCount
112:    STR R10, .WriteUnsignedNum
113:    // End line
114:    MOV R10, #newLineMsg
115:    STR R10, .WriteString
116:    //
117:    // Get codebreaker's guess
118:    BL getcode
119:    BL queryCodeToArray
120:
121:    B query
122: // out of guesses
123: break:
124:    HALT
125: //=====
126: // Continue stage 5 here
127:    query:
128:
129:    B queryloop

```

Program

```

1061 STR R10, .WriteString
1062 //
1063 // Print 'This is guess number: '
1064 MOV R10, #guessNumberCountMsg
1065 STR R10, .WriteString
1066 // Print guess number
1067 LDRB R10, currentGuessCount
1068 STR R10, .WriteUnsignedNum
1069 // End line
1070 MOV R10, #newLineMsg
1071 STR R10, .WriteString
1072 //
1073 // Get codebreaker's guess
1074 BL getCode
1075 BL queryCodeToArray
1076 B query
1077 // out of guesses
1078 break:
1079 HALT
1080 //=====
1081 // Continue stage 5 here
1082 query:
1083     B queryloop
1084 //=====
1085 // FUNCTIONS
1086 //
1087 // Stage 2 - A Code Entry Function - Vandy Aum & Marcus Mifsud
1088 //
1089 // GET CODE FUNCTION
1090 getCode:
1091 // store address of where the function was called from
1092 MOV R8, LR
1093 // branch here if the code entered is invalid
1094 getCodeNested:
1095 // Read input of code
1096 MOV R12, #tempcode
1097 // Initialize R6
1098 MOV R6, #0
1099 MOV R6, #secretcode

```

Load Save Edit

Processor

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

Input/Output

What is your guess Vandy?
This is guess number: 5
Program HALTED. STOP, LOAD or EDIT
ffff

Memory

000	0x0	0x4	0x8	0xc
0x0000	0xe300a491	0xe50fa0ec	0xe3a0a0f1	0xe50fa0b4
0x0001	0xe50fa0f8	0xe3a0a0e4	0xe50fa100	0xe3a0a0af
0x0002	0xe50fa0c8	0xe50fa10c	0xe300a4b2	0xe50fa114
0x0003	0xe51fb130	0xe50fb128	0xe300a4fe	0xe50fa124
0x0004	0xe300a524	0xe50fa12c	0xe3a0a0f8	0xe50fa134
0x0005	0xe300a44c	0xe50fa13c	0xe300a55b	0xe50fa144
0x0006	0xeb00001a	0xeb00004c	0xe3a09300	0xe5df341c
0x0007	0xe2833001	0xe5c5f3414	0xe5153000b	0xca000011
0x0008	0xe3a03000	0xe300a4c1	0xe50fa170	0xe3a0a0f1
0x0009	0xe50fa178	0xe300a55d	0xe50fa180	0xe300a55b
0x000a	0xe50fa188	0xe300a4d5	0xe50fa190	0xe50fa3dc
0x000b	0xe50fa1a4	0xe300a550	0xe50fa1a0	0xeb000003
0x000c	0xeb000037	0xea000000	0xe1000070	0xeaffffa5
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	0xeafffffe3	0xe300a583	0xe50fa230	0xeafffffe0
0x0015	0xe300a5ab	0xe50fa23c	0xeafffffd	0xe3590000
0x0016	0x0affffff7	0xe3590072	0x0a00000a	0xe3590067
0x0017	0x0a000008	0xe3590062	0x0a000006	0xe3590079
0x0018	0x0a000004	0xe3590070	0x0a000002	0xe3590063
0x0019	0x0a000000	0xeaffffe7	0xe1a0f00e	0xe3a06d0e
0x001a	0xea000001	0xe3a06001	0xeaffffff	0xe3a07000
0x001b	0xe5df314c	0xe1a07127	0xe7dc9007	0xe1a07107
0x001c	0xe7869007	0xe2877004	0xe1570003	0xbaffffff8
0x001d	0xeaffffff0	0xe3a0600e	0xea000001	0xe3a06b01
0x001e	0xeafffffff	0xe3a07000	0xe5df3114	0xe7d6a007
0x001f	0xe50fa2dc	0xe2877004	0xe1570003	0xbaffffffa

Hex Clear

ARMLite Simulator V1.3.1 © Peter Higginson 2020-24
[Documentation](#)

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.

File: stage5.txt - *comparecodes* function

```
128: comparecodes:
129:     // Initializing registers
130:     MOV R0, #0 // Case 1 Counter
131:     MOV R1, #0 // Case 2 Counter
132:     MOV R3, #0
133:     LDRB R3, arraySize // Array Size
134:     MOV R9, #0 // Query character
135:     MOV R4, #0 // Secret character
136:     MOV R5, #querycode // Query array address
137:     MOV R6, #secretcode // Secret array address
138:     MOV R7, #0 // array index / loop counter
139:     // R2 - Inner index
140:
141:     // Case 1
142:     case1start:
143:         // initialize R2 (inner index)
144:         MOV R2, #0
145:         // Load a char from query code into R9
146:         LDRB R9, [R5 + R7]
147:         //
148:         // Load a char from secret code into R4
149:         LDRB R4, [R6 + R7]
150:         //
151:         // Compare for Case 1 (BEQ)
152:         CMP R4, R9
153:         // If case 1 is true
154:         BEQ case1true
155:         // If case 1 is false
156:         B case2start
157:
158:     // Case 2
159:     case2start:
160:         // if main index = inner index, skip case2 check
161:         CMP R2, R7
162:         BEQ case2loopback
163:         // load secret char
164:         LDRB R4, [R6 + R2]
165:         // Compare secret char to query char
166:         CMP R4, R9
167:         // if case 2 is true
168:         BEQ case2true
169:         //
170:         // branch here to skip comparison of chars already done in case 1
171:         case2loopback:
172:         // increment inner index
173:         ADD R2, R2, #4
```

```

174:
175:     // loop until full array checked
176:     CMP R2, R3
177:     BLT case2start
178:     B charQueryEnd
179:
180:     //////////////////////////////////////
181:     // Case 1 success
182:     // Query char matches secret char in same position
183:     case1true:
184:         // Add 1 to case1 counter
185:         ADD R0, R0, #1
186:         B case2start
187:
188:
189:     // Case 2 success
190:     // Query char matches a secret char in a different position
191:     case2true:
192:         // Add 1 to case2 counter
193:         ADD R1, R1, #1
194:         // go back to case 2 start
195:         B case2loopback
196:
197:     // Loop back to main
198:     charQueryEnd:
199:         // Add 4 to main index
200:         ADD R7, R7, #4
201:         // Have we hit end of loop?
202:         CMP R7, R3
203:         // if not, loop back to start of char checks
204:         BLT case1start
205:         // if loop complete, display char check info
206:         B guessfeedback

```

Program

```

183 case2true:
184 // Add 1 to case2 counter
185 ADD R1, R1, #1
186 // go back to case 2 start
187 B case2loopback
188 // Loop back to main
189 charQueryEnd:
190 // Add 4 to main index
191 ADD R7, R7, #4
192 // Have we hit end of loop?
193 CMP R7, R3
194 // if not, loop back to start of char checks
195 BLT case1start
196 // if loop complete, display char check info
197 B guessfeedback
198 ///////////////////////////////////////////////////
199 //5(b)
200 guessfeedback:
201 HALT
202 // Display position matches message
203 MOV R10, #positionMatchesMsg
204 STR R10, .WriteString
205 // Display case 1 counter
206 STR R0, .WriteUnsignedNum
207 // Display colour matches message
208 MOV R10, #colourMatchesMsg
209 STR R10, .WriteString
210 // Display case 2 counter
211 STR R1, .WriteUnsignedNum
212 // If 4 position matches
213 CMP R0, #4
214 // Branch to winstate
215 BEQ winstate
216 //
217 // Initialize to currentGuessCount
218 MOV R3, #0
219 LDRB R3, currentGuessCount
220 // If this was final guess
221 CMP R3, R11

```

Load Save Edit

Processor

PC	0x00000138
LR	0x00000000
SP	0x00100000
R12	0x00000510
R11	0x00000005
R10	0x000006a3
R9	0x00000002
R8	0x000000ac
R7	0x00000010
R6	0x00000480
R5	0x00000500
R4	0x00000002
R3	0x00000010
R2	0x00000010
R1	0x00000002
R0	0x00000002

Count: 410

Current Instruction:
 Status bits: NZCV 0110

Input/Output

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

Program HALTED. STOP, LOAD or EDIT

rgyb

Memory

0x0	0x4	0x8	0xc
0x0000	0xe300a591	0xe50fa0ec	0xe3a0afd2
0x0001	0xe50fa0f8	0xe3a0ae5a	0xe50fa100
0x0002	0xe50fa0c8	0xe50fa10c	0xe300a5b2
0x0003	0xe51fb130	0xeb0000ad	0xe50fb12c
0x0004	0xe50fa128	0xe300a624	0xe50fa130
0x0005	0xe50fa138	0xe300a5ec	0xe50fa140
0x0006	0xe50fa148	0xeb000054	0xeb000086
0x0007	0xe50fa158	0xe3a0aff2	0xe50fa160
0x0008	0xe50fa168	0xe300a6a3	0xe50fa170
0x0009	0xe300a5d5	0xe50fa17c	0xe50fa190
0x000a	0xe300a6a3	0xe50fa18c	0xeb000043
0x000b	0xea000000	0xe3a00000	0xe3a01000
0x000c	0xe5df3380	0xe3a09000	0xe3a05c05
0x000d	0xe3a06d12	0xe3a07000	0xe3a02000
0x000e	0xe7d64007	0xe1540009	0xeb000009
0x000f	0xe1520007	0xeb000002	0xe7d64002
0x0010	0xeb000005	0xe2812004	0xe1520003
0x0011	0xea000003	0xe2800001	0xea000000
0x0012	0xea000000	0xe2877004	0xe1570003
0x0013	0xea000000	0xe1000070	0xe300a65b
0x0014	0xe50fb234	0xe300a65e	0xe50fa230
0x0015	0xe3500004	0xeb000004	0xe3a03000
0x0016	0xe153000b	0xeb000007	0xea000000
0x0017	0xe50fa258	0xe3a0aff2	0xe50fa260
0x0018	0xe50fa268	0xeb000006	0xe300a6a3
0x0019	0xe3a0aff2	0xe50fa27c	0xe300a68c
0x001a	0xea000000	0xeb000060	0xe300a6a3
0x001b	0xe300a698	0xe50fa29c	0xe1000070
0x001c	0xe3a0ce51	0xe3a06000	0xe3a06d12
0x001d	0xe5d69000	0xe3590000	0xeb000000
0x001e	0xe50fc2c4	0xea000001	0xe50fc290
0x001f	0xe5dc9000	0xeb000013	0xe5dc9001

Hex Clear

ARMLite Simulator V1.3.1 © Peter Higginson 2020-24
[Documentation](#)

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.

File: stage5.txt

```
209: guessfeedback:
210:     // Display position matches message
211:     MOV R10, #positionMatchesMsg
212:     STR R10, .WriteString
213:     // Display case 1 counter
214:     STR R0, .WriteUnsignedNum
215:     // Display colour matches message
216:     MOV R10, #colourMatchesMsg
217:     STR R10, .WriteString
218:     // Display case 2 counter
219:     STR R1, .WriteUnsignedNum
220:     // If 4 position matches
221:     CMP R0, #4
222:     // Branch to winstate
223:     BEQ winstate
224:     //
225:     // Initialize to currentGuessCount
226:     MOV R3, #0
227:     LDRB R3, currentGuessCount
228:     // If this was final guess
229:     CMP R3, R11
230:     BEQ losestate
231:     // Else, loop back for another guess
232:     B queryloop
233:
234: winstate:
235:     MOV R10, #newLineMsg
236:     STR R10, .WriteString
237:     MOV R10, #winStateMsg1
238:     STR R10, .WriteString
239:     MOV R10, #codeBreaker
240:     STR R10, .WriteString
241:     MOV R10, #winStateMsg2
242:     STR R10, .WriteString
243:     B gameover
244:
245: losestate:
246:     MOV R10, #newLineMsg
247:     STR R10, .WriteString
248:     MOV R10, #loseStateMsg1
249:     STR R10, .WriteString
250:     MOV R10, #codeBreaker
251:     STR R10, .WriteString
```

```

252:      MOV R10, #loseStateMsg2
253:      STR R10, .WriteString
254:      B gameover
255:
256:  gameover:
257:      // clean our words if we wish to re-run
258:      BL clean
259:      MOV R10, #newLineMsg
260:      STR R10, .WriteString
261:      MOV R10, #gameOverMsg
262:      STR R10, .WriteString
263:      HALT

```

The screenshot shows the ARMLite Simulator V1.3.1 interface. The Program window displays assembly code for a game, including a 'gameover' section. The Processor window shows the current instruction and status bits. The Memory window shows the memory dump.

Stage 5b: Screenshot showing feedback for a guess

The screenshot shows the ARMLite Simulator V1.3.1 interface. The Program window displays assembly code for a game, including a 'gameover' section. The Processor window shows the current instruction and status bits. The Memory window shows the memory dump.

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.

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

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

Appendix - Full Code Stack

```

File: mastermind.asm
001: //=====
002: // COS10031 - Computer Technology | Assessment 3
003: // Vandy Aum, Marcus Mifsud, Nicole Reichert, Luke Byrnes
004: //
005: //      _ _ _ _ _      _ _      _ _      _ _      _ _
006: //      | | \ / | | | | | | | | | | | | | | | | | | | | | | | | | |
007: //      | | \ / | | | | | | | | | | | | | | | | | | | | | | | | | |
008: //      | | \ / | | | | | | | | | | | | | | | | | | | | | | | | | |
009: //      | | \ / | | | | | | | | | | | | | | | | | | | | | | | | | |
010: //
011: // Register Assignations
012: // R0 (Compare Code of Correct Pos/Col)
013: // R1 (Compare Code of (Correct Pos, Incorrect Col))
014: // R2
015: // R3
016: // R4
017: // R5
018: // R6
019: // R7
020: // R8 Function Return (stores LR to return after a function is used within a
    function)
021: // R9 Code character address
022: // R10 String Handling
023: // R11 Guess Limit
024: // R12 Address to temp code
025: //=====
026: // Stage 1 - Game Setup - Luke Byrnes & Nicole Reichert
027: //
028: // Prompt and store Codemaker Name
029: // Set whoIsCodeMakerMsg Query prompt to R10
030: MOV R10, #whoIsCodeMakerMsg
031: // print whoIsCodeMakerMsg Query from R10
032: STR R10, .WriteString
033: // Move codeMaker address to R10
034: MOV R10, #codeMaker
035: // Take input from user and store to R10
036: STR R10, .ReadString
037: // Print codeMaker value from R10
038: STR R10, .WriteString
039: //
040: // Prompt and store CodeBreaker Name
041: // Set whoIsCodeMakerMsg Query prompt to R10
042: MOV R10, #whoIsCodeBreakerMsg
043: // print whoIsCodeMakerMsg Query from R10
044: STR R10, .WriteString
045: // Move codeBreaker address to R10
046: MOV R10, #codeBreaker
047: // Take input from user and store to R10
048: STR R10, .ReadString
049: // Print codeMaker value from R10

```

```

050:     STR R10, .WriteString
051: //
052: // Prompt and store GuessLimit for the session
053: guesslimitprompt:
054:     // Set guessLimit Query prompt to R10
055:     MOV R10, #whatIsGuessLimitMsg
056:     // Print whatIsGuessLimitMsg from R10
057:     STR R10, .WriteString
058:     // Take input from user and store to R11
059:     LDR R11, .InputNum
060:     // BL to check we have a value between 1 and 100
061:     BL guesslimitcheck
062:     // Print guessLimit from R11
063:     STR R11, .WriteUnsignedNum
064: //=====
065: // Stage 3 - A Code Entry Function - Marcus Mifsud & Vandy Aum
066: // Marcus - Consolidated into functions and split stage 2 and 3
067: // Display request for secret code entry
068:     MOV R10, #promptRuleMsg
069:     STR R10, .WriteString
070:     MOV R10, #colorChoiceMsg
071:     STR R10, .WriteString
072:     MOV R10, #codeMaker
073:     STR R10, .WriteString
074:     MOV R10, #requestCodeMsg
075:     STR R10, .WriteString
076:     MOV R10, #newlineMsg
077:     STR R10, .WriteString
078:     BL getcode
079:     BL secretCodeToArray
080: //=====
081: // Stage 4 - Query Code Entry - Nicole Reichert & Marcus Mifsud
082: // Marcus - Consolidated Nicole's work into main code base
083: queryloop:
084:     // Initialize to currentGuessCount
085:     //MOV R3, #0
086:     //LDRB R3, currentGuessCount
087:     // Increment guess count by 1
088:     //ADD R3, R3, #1
089:     //STRB R3, currentGuessCount
090:     // reset R3
091:     //MOV R3, #0
092:     //
093:     // Continue to guess now that we've checked guess count
094:     // Print 'What is your guess'
095:     MOV R10, #requestGuessMsg
096:     STR R10, .WriteString
097:     // Print codebreaker name
098:     MOV R10, #codeBreaker
099:     STR R10, .WriteString
100:     // Print question mark
101:     MOV R10, #questionMarkMsg
102:     STR R10, .WriteString

```

```

103:    // End line
104:    MOV R10, #newlineMsg
105:    STR R10, .WriteString
106:    //
107:    // Perform guessCount check now
108:    BL guesscountcheck
109:    // Print 'This is guess number: '
110:    MOV R10, #guessNumberCountMsg
111:    STR R10, .WriteString
112:    // Print guess number
113:    LDRB R10, currentGuessCount
114:    STR R10, .WriteUnsignedNum
115:    // End line
116:    MOV R10, #newlineMsg
117:    STR R10, .WriteString
118:    //
119:    // Get codebreaker's guess
120:    BL getcode
121:    BL queryCodeToArray
122:
123:
124:    B comparecodes
125:    //=====
126:    // Stage 5 - Query Code Evaluation - Nicole Reichert
127:    // 5(a)
128:    comparecodes:
129:        // Initializing registers
130:        MOV R0, #0 // Case 1 Counter
131:        MOV R1, #0 // Case 2 Counter
132:        MOV R3, #0
133:        LDRB R3, arraySize // Array Size
134:        MOV R9, #0 // Query character
135:        MOV R4, #0 // Secret character
136:        MOV R5, #querycode // Query array address
137:        MOV R6, #secretcode // Secret array address
138:        MOV R7, #0 // array index / loop counter
139:        // R2 - Inner index
140:
141:        // Case 1
142:        case1start:
143:            // initialize R2 (inner index)
144:            MOV R2, #0
145:            // Load a char from query code into R9
146:            LDRB R9, [R5 + R7]
147:            //
148:            // Load a char from secret code into R4
149:            LDRB R4, [R6 + R7]
150:            //
151:            // Compare for Case 1 (BEQ)
152:            CMP R4, R9
153:            // If case 1 is true
154:            BEQ case1true
155:            // If case 1 is false

```

```

156:      B case2start
157:
158:  // Case 2
159:  case2start:
160:      // if main index = inner index, skip case2 check
161:      CMP R2, R7
162:      BEQ case2loopback
163:      // load secret char
164:      LDRB R4, [R6 + R2]
165:      // Compare secret char to query char
166:      CMP R4, R9
167:      // if case 2 is true
168:      BEQ case2true
169:      //
170:      // branch here to skip comparison of chars already done in case1
171:      case2loopback:
172:      // increment inner index
173:      ADD R2, R2, #4
174:
175:      // loop until full array checked
176:      CMP R2, R3
177:      BLT case2start
178:      B charQueryEnd
179:
180:  //////////////////////////////////////
181:  // Case 1 success
182:  // Query char matches secret char in same position
183:  case1true:
184:      // Add 1 to case1 counter
185:      ADD R0, R0, #1
186:      B case2start
187:
188:
189:  // Case 2 success
190:  // Query char matches a secret char in a different position
191:  case2true:
192:      // Add 1 to case2 counter
193:      ADD R1, R1, #1
194:      // go back to case 2 start
195:      B case2loopback
196:
197:  // Loop back to main
198:  charQueryEnd:
199:      // Add 4 to main index
200:      ADD R7, R7, #4
201:      // Have we hit end of loop?
202:      CMP R7, R3
203:      // if not, loop back to start of char checks
204:      BLT case1start
205:      // if loop complete, display char check info
206:      B guessfeedback
207:      //////////////////////////////////////
208:  //5(b)

```

```

209: guessfeedback:
210:     // Display position matches message
211:     MOV R10, #positionMatchesMsg
212:     STR R10, .WriteString
213:     // Display case 1 counter
214:     STR R0, .WriteUnsignedNum
215:     // Display colour matches message
216:     MOV R10, #colourMatchesMsg
217:     STR R10, .WriteString
218:     // Display case 2 counter
219:     STR R1, .WriteUnsignedNum
220:     // If 4 position matches
221:     CMP R0, #4
222:     // Branch to winstate
223:     BEQ winstate
224:     //
225:     // Initialize to currentGuessCount
226:     MOV R3, #0
227:     LDRB R3, currentGuessCount
228:     // If this was final guess
229:     CMP R3, R11
230:     BEQ losestate
231:     // Else, loop back for another guess
232:     B queryloop
233:
234: winstate:
235:     MOV R10, #newLineMsg
236:     STR R10, .WriteString
237:     MOV R10, #winStateMsg1
238:     STR R10, .WriteString
239:     MOV R10, #codeBreaker
240:     STR R10, .WriteString
241:     MOV R10, #winStateMsg2
242:     STR R10, .WriteString
243:     B gameover
244:
245: losestate:
246:     MOV R10, #newLineMsg
247:     STR R10, .WriteString
248:     MOV R10, #loseStateMsg1
249:     STR R10, .WriteString
250:     MOV R10, #codeBreaker
251:     STR R10, .WriteString
252:     MOV R10, #loseStateMsg2
253:     STR R10, .WriteString
254:     B gameover
255:
256: gameover:
257:     // clean our words if we wish to re-run
258:     BL clean
259:     MOV R10, #newLineMsg
260:     STR R10, .WriteString
261:     MOV R10, #gameOverMsg

```

```

262:         STR R10, .WriteString
263:         HALT
264:
265: //=====
266: // FUNCTIONS
267: //
268: // Stage 2 - A Code Entry Function - Vandy Aum & Marcus Mifsud
269: //
270: // GET CODE FUNCTION
271: getcode:
272:     // store address of where the function was called from
273:     MOV R8, LR
274:     // branch here if the code entered is invalid
275:     getcodeNested:
276:         // Read input of code
277:         MOV R12, #tempcode
278:         // Initialize R6
279:         MOV R6, #0
280:         MOV R6, #secretcode
281:         MOV R9, #0
282:         LDRB R9, [R6]
283:         CMP R9, #0
284:         BEQ secretcodeentry
285:         BNE querycodeentry
286:         // If codemaker's turn
287:         secretcodeentry:
288:             STR R12, .ReadSecret
289:             B validateCharLoop
290:         // If codebreaker's turn
291:         querycodeentry:
292:             STR R12, .ReadString
293:             B validateCharLoop
294:         // Validate Secret Code
295:         validateCharLoop:
296:         // First Character
297:             // Store the address of the first byte of R12 content (temp
                code) in R9
298:             LDRB R9, [R12]
299:             BL validateChar
300:         // Second Character
301:             // Store the address of the second byte of R12 content (temp
                code) in R9
302:             //one character is one byte so when adding one byte to R12
                it will be the address of the next character
303:             LDRB R9, [R12, #1]
304:             BL validateChar
305:         // Third Character
306:             // Store the address of the third byte of R12 content (temp
                code) in R9
307:             LDRB R9, [R12, #2]
308:             BL validateChar
309:         // Fourth Character
310:             // Store the address of the fourth byte of R12 content (temp

```

```

        code) in R9
311:        LDRB R9, [R12, #3]
312:        BL validateChar
313:        // Fifth Character
314:        // Store the address of the fifth byte of R12 content (temp
        code) in R9
315:        LDRB R9, [R12, #4]
316:        CMP R9, #0        //check if a character was not entered
317:        BNE overLimit    //if a character was entered branch to
                           'overLimit'
318:        //if a fifth character was not entered and all prior checks
        passed, input is valid, return to code
319:        // return address the function was called from to LR
320:        MOV LR, R8
321:        B Return
322:
323: invalidChar:
324:        MOV R10, #errorMsg1
325:        STR R10, .WriteString
326:        b getcodeNested
327: tooFewChar:
328:        MOV R10, #errorMsg2
329:        STR R10, .WriteString
330:        b getcodeNested
331: overLimit:
332:        MOV R10, #errorMsg3
333:        STR R10, .WriteString
334:        b getcodeNested
335:
336: // VALIDATE CHARACTER FUNCTION
337: validateChar:
338:        CMP R9, #0        //check if a character was not entered
339:        BEQ tooFewChar
340:        CMP R9, #0x72    //check if the character is r(red)
341:        BEQ Return
342:        CMP R9, #0x67    //check if the character is g(green)
343:        BEQ Return
344:        CMP R9, #0x62    //check if the character is b(blue)
345:        BEQ Return
346:        CMP R9, #0x79    //check if the character is y(yellow)
347:        BEQ Return
348:        CMP R9, #0x70    //check if the character is p(purple)
349:        BEQ Return
350:        CMP R9, #0x63    //check if the character is c(cyan)
351:        BEQ Return
352:        b invalidChar    //branch to 'invalidChar' if the character was not
                           matched by any of the above checks
353:
354: // Function to return from function
355: Return: RET
356:
357: // STORE CODE TO ARRAY FUNCTION
358: // R12 - Address to tempcode is stored here

```



```

359: // R9 - Current Character
360: // R6 - Memory address of the array to fill
361: // R7 - Array index
362: // R3 - Array Size
363: secretCodeToArray:
364:     // load the address of the secret code into R6
365:     MOV R6, #secretcode
366:     B codeToArray
367: queryCodeToArray:
368:     MOV R6, #querycode
369:     B codeToArray
370: codeToArray:
371:     // initialize the array position to 0
372:     MOV R7, #0
373:     // initialize array size
374:     LDRB R3, arraySize
375:     fillArrayLoop:
376:         // divide R7 (index) by 4
377:         LSR R7, R7, #2
378:         // load character into R9
379:         LDRB R9, [R12 + R7]
380:         // multiply R7 (index) by 4
381:         LSL R7, R7, #2
382:
383:         // store character into array element
384:         STR R9, [R6 + R7]
385:
386:         // increment index counter by 4
387:         ADD R7, R7, #4
388:
389:         CMP R7, R3 // repeat until 4 elements of the array have been
                        filled
390:         BLT fillArrayLoop
391:     B Return
392:
393: // OUTPUT ARRAY FUNCTION (only used for testing)
394: // output secret code
395: outSecretCode:
396:     MOV R6, #secretcode
397:     B outCodeArray
398: // output query code
399: outQueryCode:
400:     MOV R6, #querycode
401:     B outCodeArray
402: outCodeArray:
403:     // initialize index counter
404:     MOV R7, #0
405:     // initialize array size
406:     LDRB R3, arraySize
407:     // output the 4 digit code from an array
408:     outCodeArrayLoop:
409:         LDRB R10, [R6 + R7]
410:         STR R10, .WriteChar

```

```

411:
412:     // increment index
413:     ADD R7, R7, #4
414:
415:     // loop until 4 elements have been output
416:     CMP R7, R3
417:     BLT outCodeArrayLoop
418:     B Return
419:
420: //=====
421: //HELPER FUNCTIONS - Nicole Reichert
422: // Checking bounds of functions - we could be inputting something over a
    word or over a hard limit of 1-255.
423: guesslimitcheck:
424: CMP R11, #0xFF
425: BGT exceedlimit
426: CMP R11, #0x1
427: BLT exceedlimit
428: RET
429:
430: exceedlimit:
431: MOV R10, #errorMsg4
432: STR R10, .WriteString
433: MOV R11, #0
434: B guesslimitprompt
435:
436: guesscountcheck:
437: // Initialize to currentGuessCount
438:     MOV R3, #0
439:     LDRB R3, currentGuessCount
440:     // Increment guess count by 1
441:     ADD R3, R3, #1
442:     STRB R3, currentGuessCount
443:     // reset R3
444:     MOV R3, #0
445:     RET
446: clean:
447: // wipe currentguesscount
448:     LDRB R8, currentGuessCount
449:     MOV R8, #0
450:     STRB R8, currentGuessCount
451:     MOV R8, #secretcode
452:     MOV R7, #0
453:     STR R7, [R8]
454:
455:     RET
456:
457: //=====
458: // STORAGE =====
459: // Store block of memory of 128 bytes to store the codemaker's name
460: codeMaker: .BLOCK 128
461: // Store block of memory of 128 bytes to store the codebreaker's name
462: codeBreaker: .BLOCK 128

```

```

463: // Array Size
464: arraySize: .BYTE 16 // 4 elements * 4 bytes
465: // secret code array
466: .ALIGN 128
467: secretcode: .BYTE 0
468:             0
469:             0
470:             0
471: //
472: // query code array
473: .ALIGN 128
474: querycode: .BYTE 0
475:            0
476:            0
477:            0
478: //
479: // temp code string
480: tempcode: .BLOCK 128
481: //
482: currentGuessCount: .BYTE 0
483: //
484: // MESSAGES =====
485: // Display whoIsCodeMakerMsg Query prompt:
486: whoIsCodeMakerMsg: .ASCIZ "Codemaker is: "
487: // Display whoIsCodeBreakerMsg Query prompt:
488: whoIsCodeBreakerMsg: .ASCIZ "\nCodebreaker is: "
489: // Display guessLimit Query prompt:
490: whatIsGuessLimitMsg: .ASCIZ "\nGuess Limit: "
491: requestGuessMsg: .ASCIZ "What is your guess "
492: guessNumberCountMsg: .ASCIZ "This is guess number: "
493: //Display the prompt for user to input the secret code
494: requestCodeMsg: .ASCIZ " enter the Code: "
495: promptRuleMsg: .ASCIZ "\nType in 4 colors for the secret code"
496: colorChoiceMsg: .ASCIZ "\n(r-RED, g-GREEN, b-BLUE, y-YELLOW, p-PURPLE,
                        c-CYAN)\n"
497: // Outcome of guess messages
498: positionMatchesMsg: .ASCIZ "Position matches: "
499: colourMatchesMsg: .ASCIZ ", Colour matches: "
500: // Win/Lose States/GameOver
501: winStateMsg1: .ASCIZ "Exact code match.\n"
502: winStateMsg2: .ASCIZ ", you WIN!"
503: loseStateMsg1: .ASCIZ "Out of guesses.\n"
504: loseStateMsg2: .ASCIZ ", you LOSE!"
505: gameOverMsg: .ASCIZ "Game Over!"
506: // General use Messages
507: newLineMsg: .ASCIZ "\n"
508: questionMarkMsg: .ASCIZ "?"
509: // Error Messages
510: errorMsg1: .ASCIZ "\nError: Invalid character entered!\n"
511: errorMsg2: .ASCIZ "\nError: Not enough characters entered!\n"
512: errorMsg3: .ASCIZ "\nError: Too many characters entered!\n"
513: errorMsg4: .ASCIZ "\n--Please enter a value from 1 to 255!--\n"
514: // Test Message

```

```
515: testMsg: .ASCIZ "\nTEST\n"
```