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

Program Overview

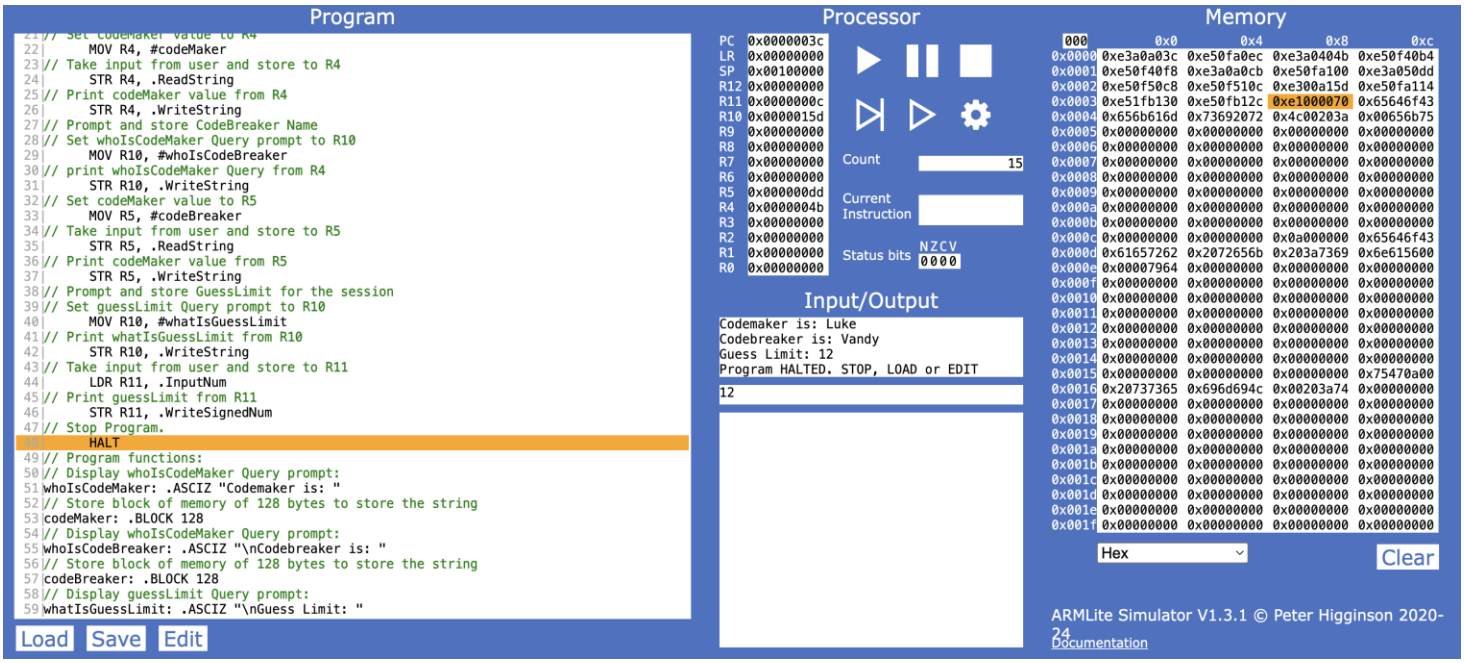
This program replicates gameplay of the Mastermind boardgame in Assembly using the ARMLite assembly utility.

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: .ASCIZ "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: .ASCIZ "\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: .ASCIZ "\nGuess Limit: "
66:     // Store the guesslimit as a label, 0 as default
67:     guessLimit: .WORD 0
```



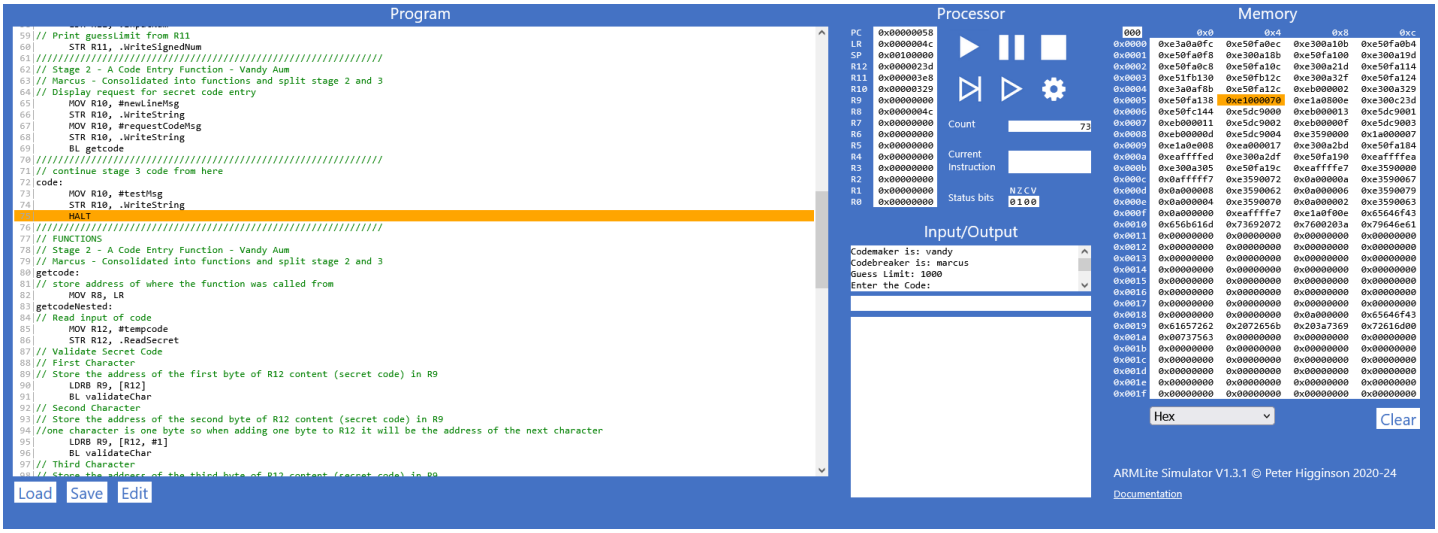
Stage 1: Functional Screenshot

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
```



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.

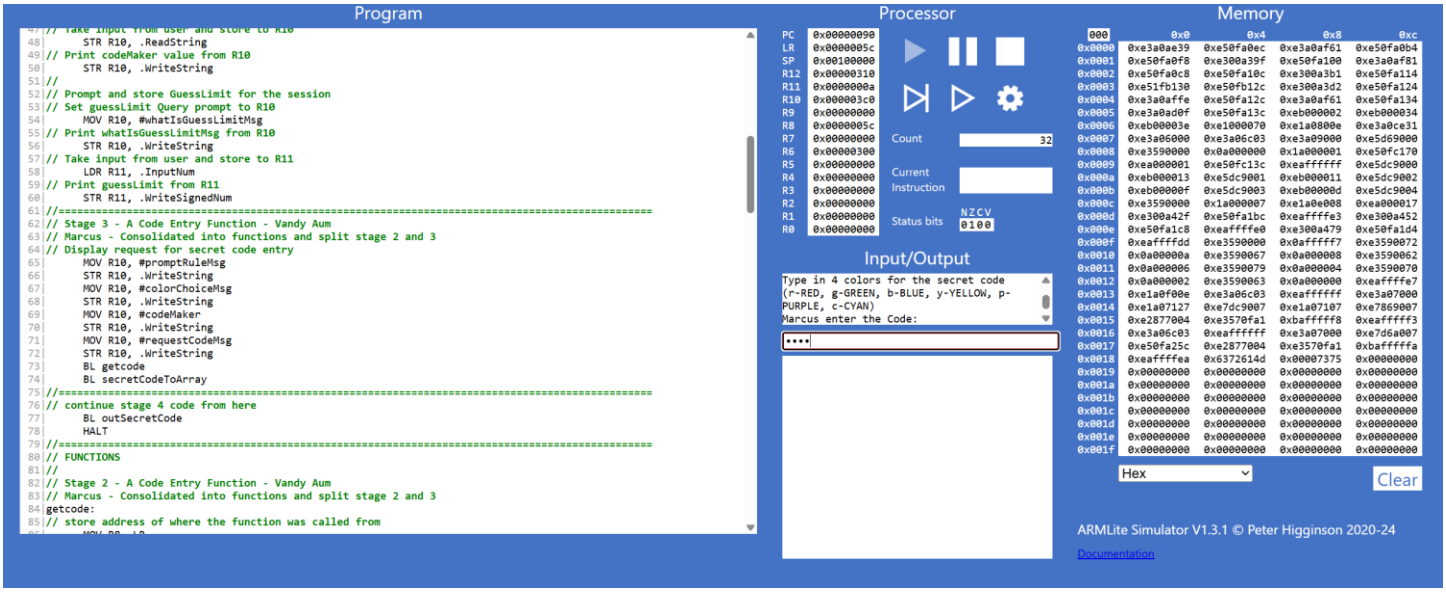
File: stage3.txt – code to array function

```
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

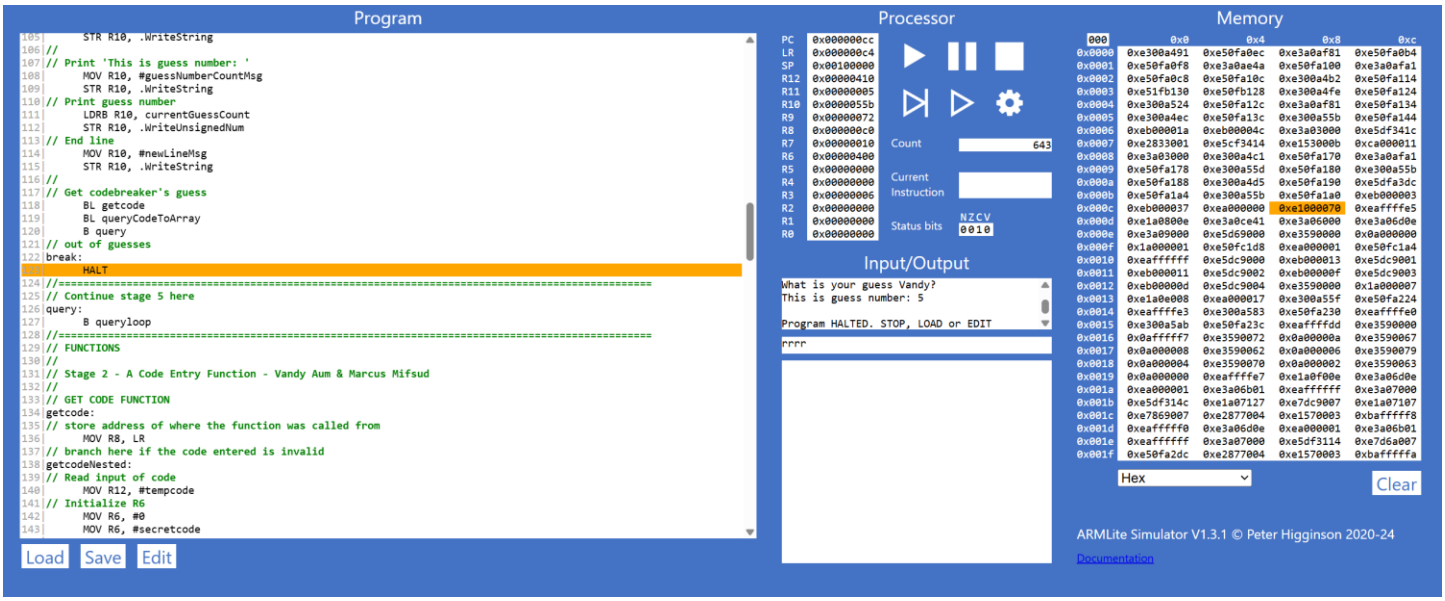
Stage 4 (stage4.txt)

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
```



Stage 4: Functional Screenshot

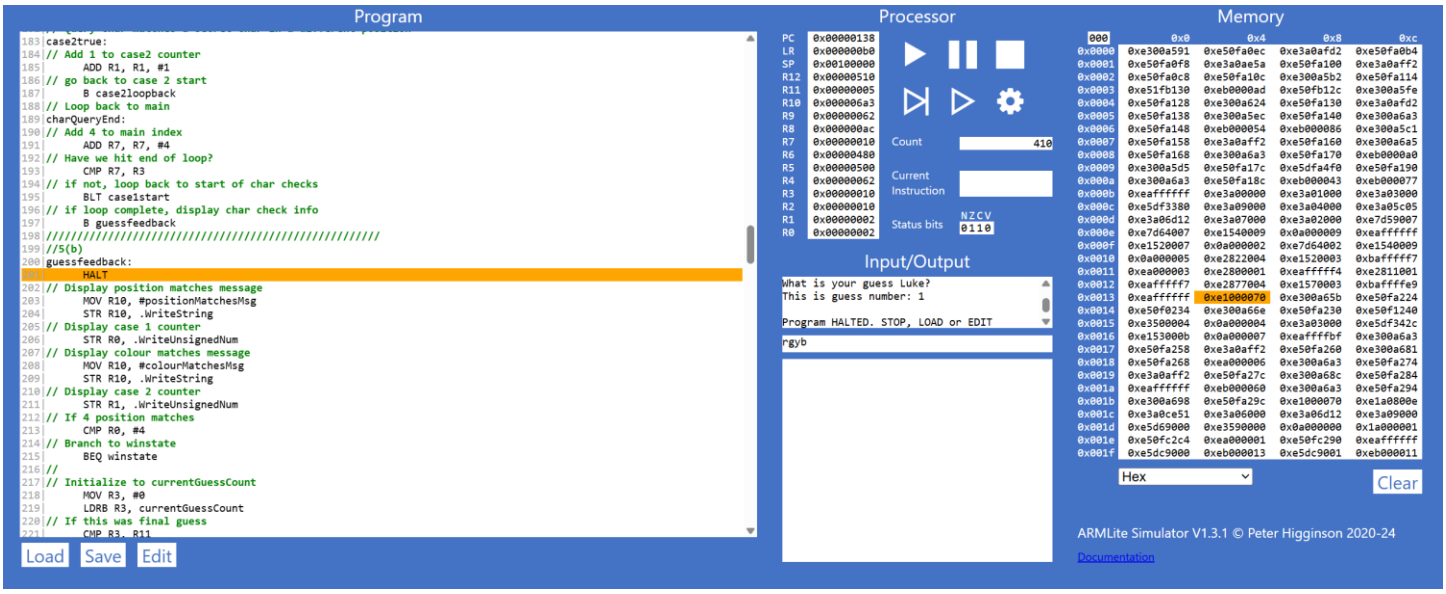
Stage 5a (stage5a.txt)

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
```



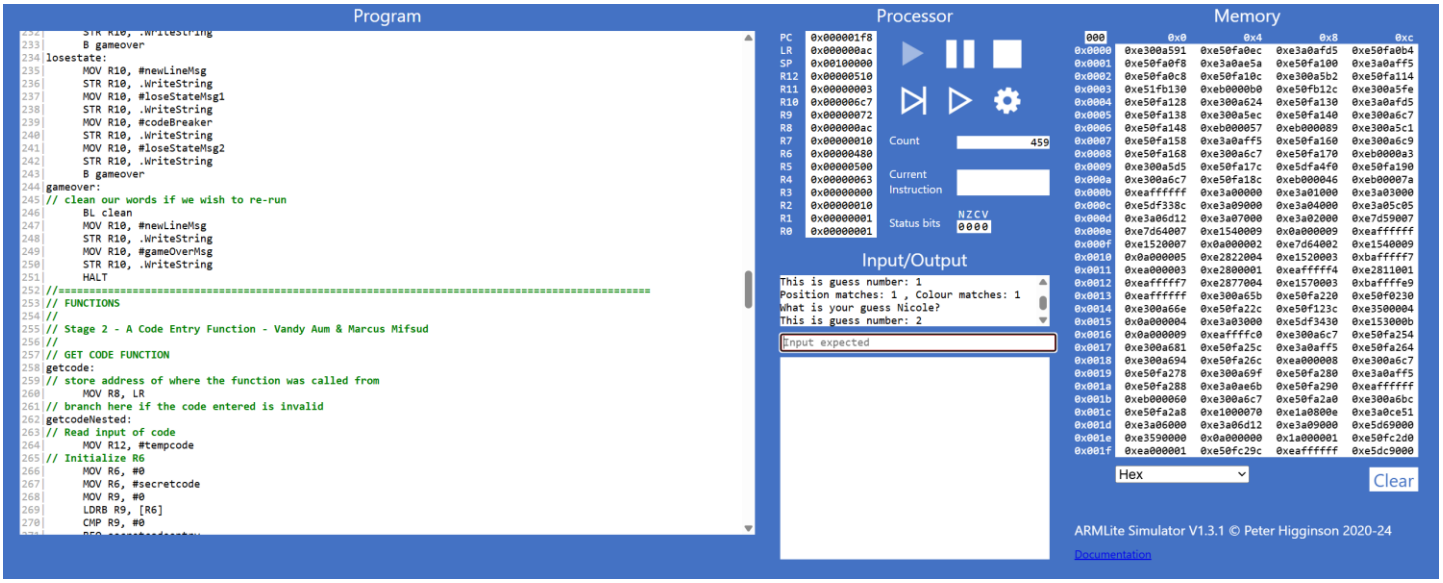
Stage 5a: Screenshot showing 2 exact matches and 2 colour matches (stored in R0 & R1)

Stage 5b (stage5b.txt)

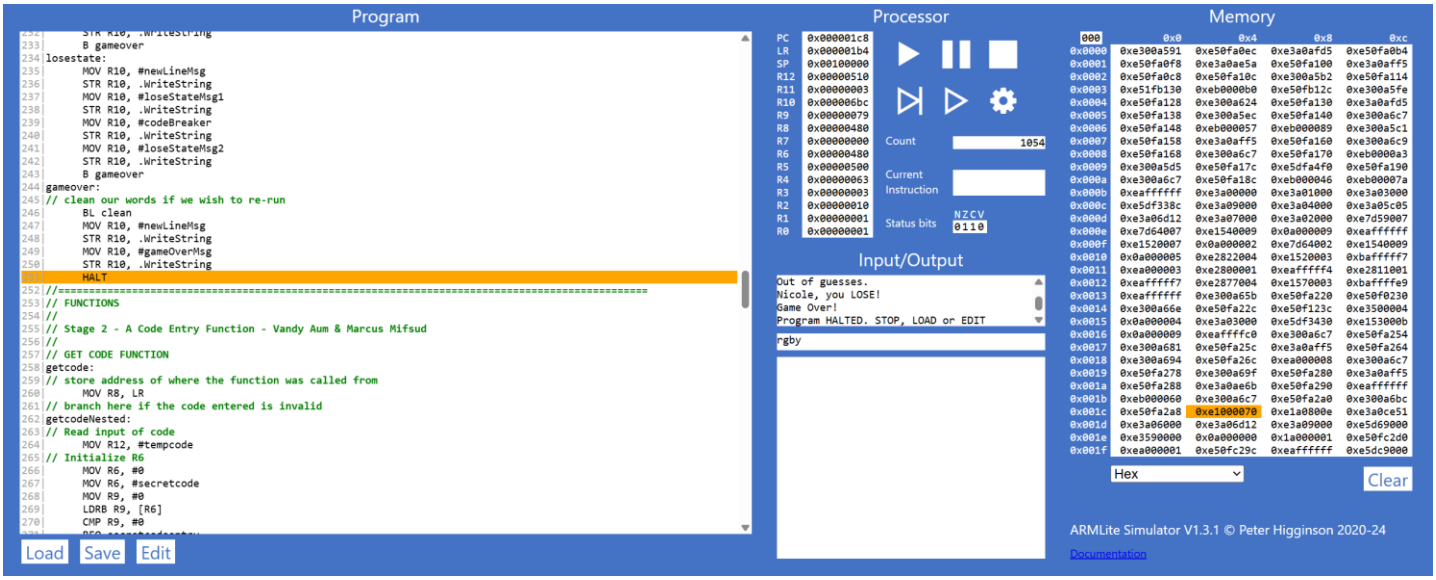
In stage 5 the guessfeedback function was created which displays the result of comparecodes. If the result of case 1 is 4 (the codes fully match) the code branches to winstate which display's a win message and then branches to gameover which ends the game. The logic for incrementing the current guess 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
```



Stage 5b: Screenshot showing feedback for a guess



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
            LDRB R9, [R12, #3]
            BL validateChar
        // Fifth Character
            // Store the address of the fifth byte of R12 content (temp
              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 to code
            // 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
    BEQ tooFewChar
    CMP R9, #0x72     //check if the character is r(red)
    BEQ Return
    CMP R9, #0x67     //check if the character is g(green)
    BEQ Return
    CMP R9, #0x62     //check if the character is b(blue)
    BEQ Return
    CMP R9, #0x79     //check if the character is y(yellow)
    BEQ Return
    CMP R9, #0x70     //check if the character is p(purple)
    BEQ Return
    CMP R9, #0x63     //check if the character is c(cyan)
    BEQ Return
    b invalidChar     //branch to 'invalidChar' if the character was not
                        matched by any of the above checks

// Function to return from function
Return: RET

// STORE CODE TO ARRAY FUNCTION
// 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:     // intialize 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"
```