Assignment 2 Report

Name: Bhuvan Virmani

Student ID: 103847174

Unit Code: COS10004

Lab Session: Thursday 10:30-12:30

DESCRIPTION: -

In this assignment I have made a game in Armlite Assembly Programming Language. In this the game asks for the name of Codemaker and a codebreaker from the user and stores them in a label, and it takes the input from the user of the number of guesses made by the codebreaker and stores It in a register. Further the Code is checked to see if it has correct characters and number of characters. After that the codebreaker is asked for its guesses and is compared with the original code. If the codebreaker matches the code he wins the game, otherwise codemaker wins the game.

STAGE 1:

```
      MOV R0, #str1
      STR R0, .WriteString
      MOV R1, #codemaker
      STR R1, .ReadString
      MOV R2, #str2
      STR R2, .WriteString
      MOV R3, #codebreaker
      STR R3, .ReadString
      MOV R4, #str3
      STR R4, .WriteString
      LDR R5, .InputNum
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

In stage 1 the first section of my code is used to display the instructions to the user to enter codemaker and codebreakers name and enter the number of guesses for the codebreaker and stores the names in the label and also stores the guesses in a register.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      MOV R0, #str4
      STR R0, .WriteString
      STR R1, .WriteString
      MOV R2, #str5
      STR R2, .WriteString
      STR R3, .WriteString
      MOV R4, #str6
      STR R4, .WriteString
      STR R5, .WriteUnsignedNum
      HALT
```

Input/Output

```
Codemaker is: Bhuvan
Codebreaker is: Sam
Maximum Number of Guesses: 5
Program HALTED. STOP, LOAD or EDIT
5
```

The second section of my code prints out the codemakers and codebreaker's name and also the number of guesses made by the codebreaker.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
str1: .ASCIZ "Enter Codemaker's Name: \n"
str2: .ASCIZ "Enter Codebreakers's Name:\n"
str3: .ASCIZ "Enter the number of guesses: \n"
str4: .ASCIZ "\nCodemaker is: "
str5: .ASCIZ "\nCodebreaker is: "
str6: .ASCIZ " \nMaximum Number of Guesses:"
```

STAGE 2:

```
      MOV R0,#code
getcode:
      MOV R1, R0        ;; passing code array as a perimeter
      MOV R2, #str7
      STR R2, .WriteString
      STR R1, .ReadString
```

In stage 2, getcode function is used to input the secret code by the codemaker and stored in a label #code.

```
testchar:               ;; testing number of characters - 4
      LDRB R4, [R1+R2]
      ADD R2,R2,#1
      CMP R4 ,R3
      BNE testchar
      CMP R2, #5
      BNE getcode
```

The "testchar" function is used to test the number of characters in the code. If the number of characters in the secret code is not equal to 4 the user must input the code again.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      MOV R2, #test
      MOV R3, #0
      MOV R5, #0x00
testASCII:              ;; testing valid characters (r,g,y,b,p,c)
      MOV R1, R0        ;; passing code array as a perimeter
      PUSH {R6,R7}
loop1:
      LDRB R6, [R1+R3]
      ADD R3,R3,#1
      MOV R4, #0
      CMP R6,R5         ;; compared with null
      BEQ continue
loop2:
      LDRB R7, [R2+R4]
      ADD R4, R4,#1
      CMP R7,R5         ;; compared with null
      BEQ getcode
      CMP R6,R7
      BNE loop2
      CMP R6,R7
      BEQ loop1
      B loop1
      POP {R6,R7}
continue:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

The "testASCII" function is used to test that the code only contains the valid characters which are r,g,b,y,p,c. Each byte (character) of the secretcode entered by the user is loaded in a register (here R6) and then compared with the valid characters loaded in another register (here R7) from a label called "test". If a Character from the secretcode matches the valid character list, it moves on to the next character (in loop1) until the register(R6) is loaded with the null it continues the code to the next part.

If the character from the secret code doesn't match the valid character list(test) and the valid character register (R7) reaches null (meaning no match), the user is prompted to enter the code again.

Basically, loop1 is used to iterate through the characters of secretcode and load them in R6. Loop2 is used to iterate through the characters in "test" label (string with valid characters) and store them In R7. Then R6 and R7 are being compared for testing.

STAGE 3:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      MOV R0,#secretcode
      MOV R2, #str7
      STR R1, .WriteString
      STR R2, .WriteString
getcode:
      MOV R1, R0        ;; passing code array as a perimeter
      STR R1, .ReadString
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

In this stage, getcode function is modified to display the codemakers name entered by the user followed with "please enter a 4-character secret code" (stored in label str7) and store the code in a label called "secretcode".

STAGE 4:

```
continue:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      MOV R1, #codebreaker
      MOV R2, #str8
      MOV R3,#0
      MOV R4, #querycode
      MOV R5, R12
guessinput:
      STR R11, .WriteChar ;;newline
      STR R1, .WriteString
      STR R2, .WriteString
      STR R5, .WriteUnsignedNum
      STR R4, .ReadString
      SUB R5, R5, #1
      CMP R5, R3
      BNE guessinput
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

In this stage, the Codebreakers name is printed followed with "this is guess number: <N>" (stored in label str8), where N is the number of guesses left. The "guessinput" function iterates through the number of guesses and in each iteration, it takes an input from the user and stores the input in the label "querycode".

STAGE 5A:

```
guessinput:
      STR R11, .WriteChar
      STR R1, .WriteString
      STR R2, .WriteString
      STR R5, .WriteUnsignedNum
      STR R4, .ReadString
      SUB R5, R5, #1
      BL comparecodes
      CMP R5, R3
      BNE guessinput
      CMP R5,R3
      BEQ gameover
```

The Function “guessinput” was modified so that each time the codebreaker guesses the code it goes the function “comparecodes” for validation of CASE 1 and CASE 2 (as described in the assignment) until the number of guesses are 0 then the function branches to gameover where the winner or loser is decided.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
comparecodes:
      PUSH {R1,R2,R3}
      PUSH {R4,R5,R6,R7,R11}
      MOV R1, #querycode
      MOV R4, R0        ;; passing secretcode array as a perimeter
      MOV R5, R1        ;; passing querycode array as a perimeter
      MOV R2, #0
      MOV R3, #0
      MOV R12,#0        ;;for counting instances of case 1
      MOV R11,#0        ;;for counting instances of case 2
      MOV R9,#0x00      ;;null value
case1:
      LDRB R6,[R4+R2]
      LDRB R7,[R5+R3]
      ADD R2,R2,#1
      ADD R3,R3,#1
      CMP R6,R9         ;;comparing with null value
      BEQ continue2
      CMP R6,R7         ;;comparing codes
      BNE case1
      PUSH {LR}
      BL case1count
      POP {LR}
      CMP R6,R9
      BNE case1
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

The function “comparecodes” takes both the codes (secretcode and querycode)as parameters. In the Label Case 1, R6 is filled with the first character of secret code and R7 is

filled with the first character of querycode and they are compared for a match until the registers are filled with null value. If a match occurs this function branches to "case1count".

```
case1count:             ;; to count the cases in case1
        ADD R12,R12,#1
        RET
```

Here the instance of position match is recorded by adding 1 to register 12. After the registers reach a null value the function is branched to "continue2" to validate for case 2.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
continue2:
        MOV R2, #0
case2:
        MOV R5, R0          ;; passing secretcode array as a perimeter
        MOV R6, R1          ;; passing querycode array as a perimeter
        LDRB R7,[R5+R2]     ;;taking a single character from secret code
        ADD R2,R2,#1
        MOV R3, #0
        CMP R7,R9           ;;comparing with null
        BEQ continue3
loop3:
        LDRB R8, [R6+R3]    ;;taking a single character from query code
        ADD R3, R3,#1
        CMP R8,R9           ;;comparing with null
        BEQ case2
        CMP R7,R8           ;;comparing codes
        BNE loop3
        PUSH {LR}
        BL case2count       ;;counting the instances of color match
        POP {LR}
        B case2
continue3:
        POP {R4,R5,R6,R7,R11}
        POP {R1,R2,R3}
        RET
gameover:
```

For case2 we take the secretcode and querycode as parameters. We load a single character of secret code into R7 and then compare it with every character taken from query code. We do this by loading a single character from query code into R8 and then iterate through the querycode (using "loop3") until the null value is reached or a match has been made. If the null value is reached we go back to case 2 and load the second character of secret code into R7 and then again compare it with the querycode.

If a match is made, we branch into "case2count" to record the instance.

```
case2count:
        ADD R11,R11,#1
        RET
```

In case2count the instance of a colour match is recorded in R11 by adding 1.

When R7 is filled with null character(meaning we have compared every character of secretcode with each and every character of querycode) we move onto continue3.

STAGE 5B:

```
138 continue3:
139         POP {R4,R5,R6,R7}
140         POP {R1,R2,R3}
141 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Stage 5b
142         MOV R8, #str9
143         MOV R9, #str10
144         STR R8, .WriteString ;;printing the output for position matches
145         STR R12, .WriteUnsignedNum
146         STR R9, .WriteString ;;printing the output for color matches
147         STR R11, .WriteUnsignedNum
148         MOV R8, #4
149         CMP R12, R8
150         BEQ win
151         RET
152 gameover:
153         MOV R2, #4
154         CMP R12, R2
155         BEQ win
156 lose:
157         STR R10, .WriteChar ;;newline char
158         STR R1, .WriteString ;;codebreaker name
159         MOV R3, #str12
160         STR R3, .WriteString
161         B over
162 win:
163         STR R10, .WriteChar ;;newline char
164         STR R1, .WriteString ;;codebreaker name
165         MOV R3, #str11
166         STR R3, .WriteString
167 over:
168         MOV R1,#str13
169         STR R1, .WriteString
170         HALT
```

```
185 str9: .ASCIZ "\nPosition Matches: "
186 str10: .ASCIZ "\nColor Matches: "
187 str11: .ASCIZ ",you WIN!"
188 str12: .ASCIZ ",you LOSE!"
189 str13: .ASCIZ "\nGAME OVER!!"
```

From line 142 to 147 we are just printing the output for the validation that we did in stage 5A (of recording the instances of CASE 1 and CASE 2). After that we are comparing R12 (which has the number of instances of position matches, Case1) with 4. So, if we have 4 position matches the code jumps to the win label where it prints that the codebreaker has won the game.

If not it returns back to the guessinput function where codemaker makes another guess, then again that guess goes through comparecodes to check for CASE 1 & CASE 2.

```
guessinput:
      STR R10, .WriteChar
      STR R1, .WriteString
      STR R2, .WriteString
      STR R5, .WriteUnsignedNum
      STR R4, .ReadString
      SUB R5, R5, #1
      BL comparecodes
      CMP R5, R3
      BNE guessinput
      CMP R5,R3         ;;comparing remaining guesses with 0
      BEQ gameover
```

If the codemaker reaches to 0 guesses remaining the function jumps to gameover if the position matches(case1count) are not equal to 4 codebreaker loses the game.

Assumption: -

I have assumed that the querycode entered by the codebreaker is valid as it doesn't goes through the validation.