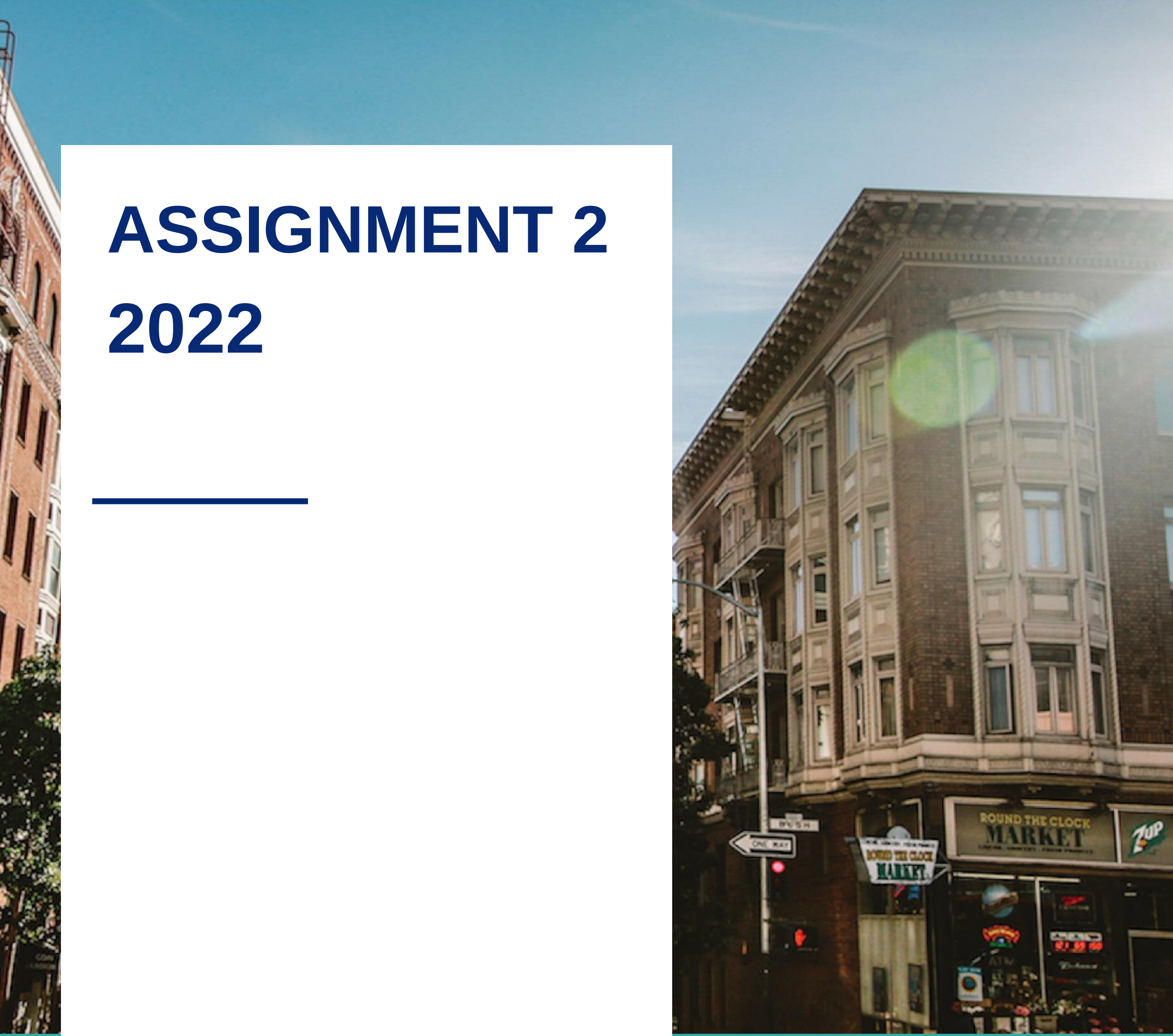


ASSIGNMENT 2 2022

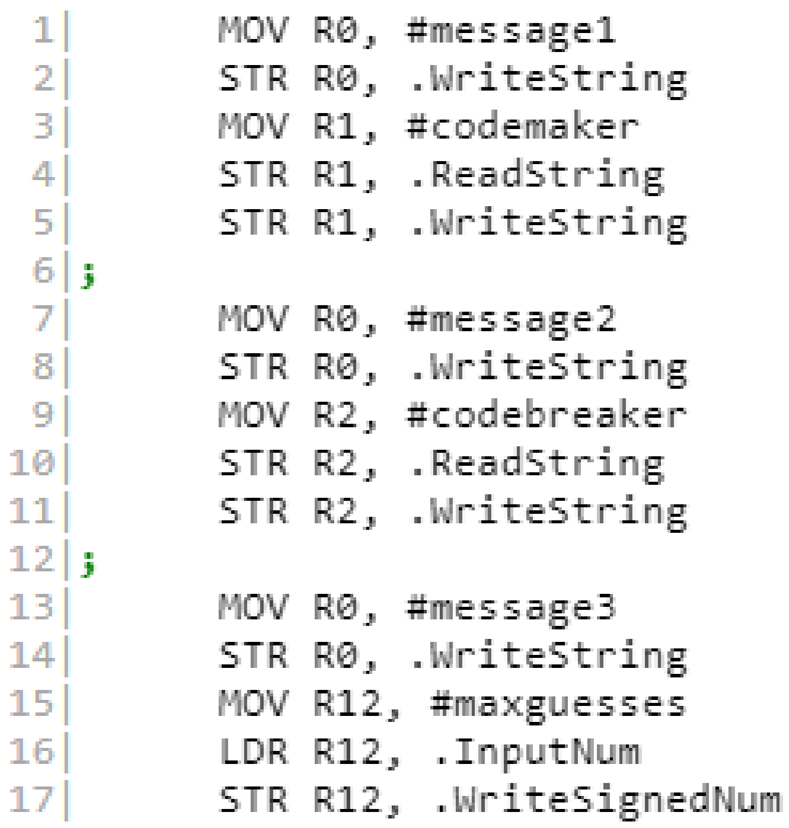
Nguyen Nam Tung
103181157
Lab Session: Thursday BA405 8h30
COS10004 Computer Systems

1. Table of content

2. Design outline

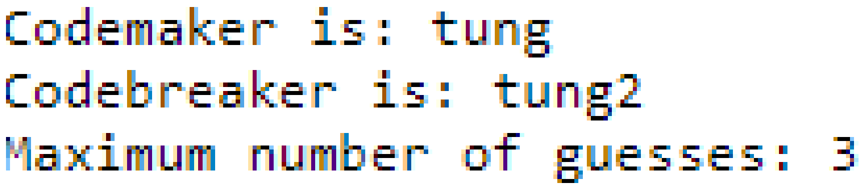
For the design outline of my program, I have submitted a total of 6 code files for each stage from stage 1 to stage 5a, and a full code program file (similar to the stage5a)

2.1. Stage 1



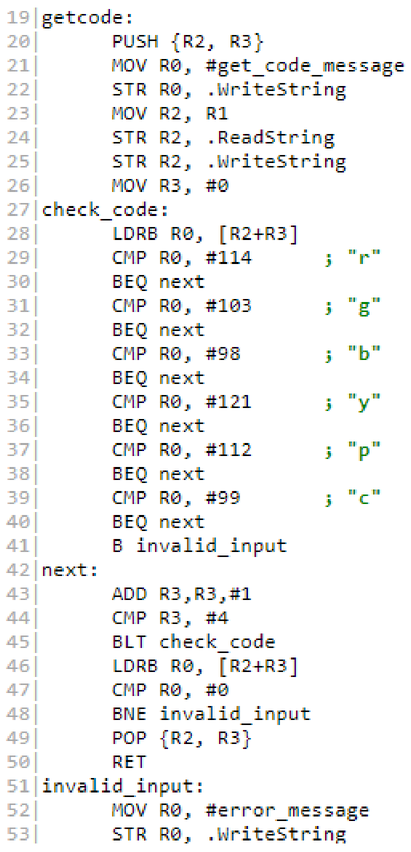
```
        MOV R0, #message1
        STR R0, .WriteString
        MOV R1, #codemaker
        STR R1, .ReadString
        STR R1, .WriteString
;
        MOV R0, #message2
        STR R0, .WriteString
        MOV R2, #codebreaker
        STR R2, .ReadString
        STR R2, .WriteString
;
        MOV R0, #message3
        STR R0, .WriteString
        MOV R12, #maxguesses
        LDR R12, .InputNum
        STR R12, .WriteSignedNum
```

I stored the required strings: `Codemaker is:` , `Codebreaker is:` , `Maximum number of guesses:` to variables (message1, message2, message3) and then I stored the input of the user into 3 variables: codemaker, codebreaker, maxguesses. Here is an example of the output:

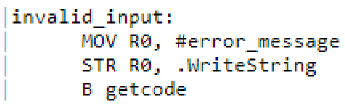


```
Codemaker is: tung
Codebreaker is: tung2
Maximum number of guesses: 3
```

2.2. Stage 2



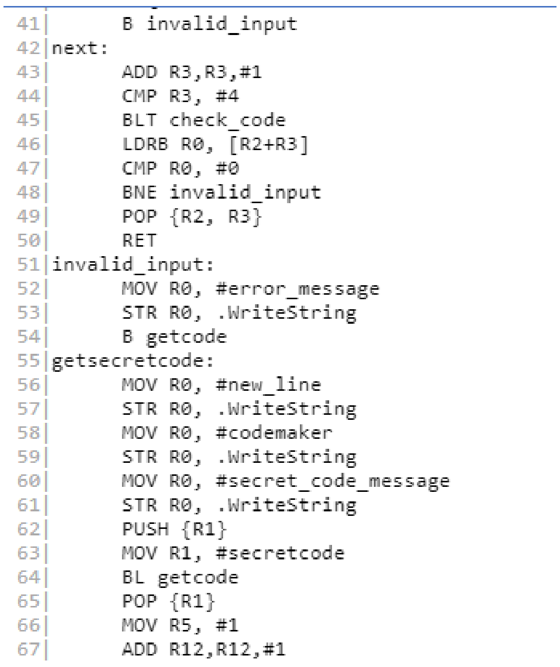
```
getcode:
      PUSH {R2, R3}
      MOV R0, #get_code_message
      STR R0, .WriteString
      MOV R2, R1
      STR R2, .ReadString
      STR R2, .WriteString
      MOV R3, #0
check_code:
      LDRB R0, [R2+R3]
      CMP R0, #114      ; "r"
      BEQ next
      CMP R0, #103      ; "g"
      BEQ next
      CMP R0, #98       ; "b"
      BEQ next
      CMP R0, #121      ; "y"
      BEQ next
      CMP R0, #112      ; "p"
      BEQ next
      CMP R0, #99       ; "c"
      BEQ next
      B invalid_input
next:
      ADD R3,R3,#1
      CMP R3, #4
      BLT check_code
      LDRB R0, [R2+R3]
      CMP R0, #0
      BNE invalid_input
      POP {R2, R3}
      RET
invalid_input:
      MOV R0, #error_message
      STR R0, .WriteString
```



```
invalid_input:
      MOV R0, #error_message
      STR R0, .WriteString
      B getcode
```

In this stage, I created a function call getcode to get the input code from the codemaker. The function behind this is to loop through all the index in the input string of the user and then compare the character in the string to the color string such as r, g. After a character is checked if the character is not valid, invalid message will show up. If it is valid, the next character in the string will be checked and the index of the loop will be incremented by 1 until all 4 characters are checked. If there is another character, an invalid message will also show up.

2.3. Stage 3

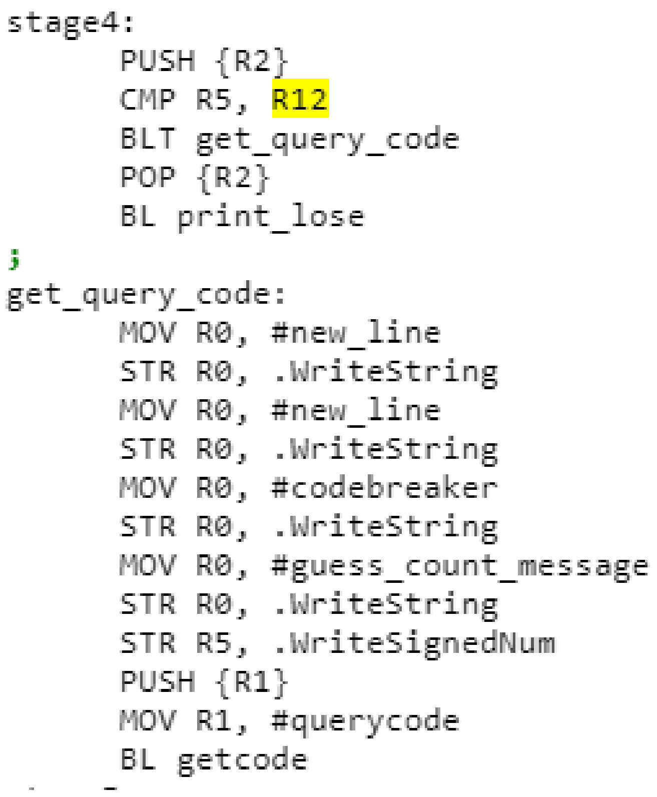


```
41|       B invalid_input
42|next:
43|       ADD R3,R3,#1
44|       CMP R3, #4
45|       BLT check_code
46|       LDRB R0, [R2+R3]
47|       CMP R0, #0
48|       BNE invalid_input
49|       POP {R2, R3}
50|       RET
51|invalid_input:
52|       MOV R0, #error_message
53|       STR R0, .WriteString
54|       B getcode
55|getsecretcode:
56|       MOV R0, #new_line
57|       STR R0, .WriteString
58|       MOV R0, #codemaker
59|       STR R0, .WriteString
60|       MOV R0, #secret_code_message
61|       STR R0, .WriteString
62|       PUSH {R1}
63|       MOV R1, #secretcode
64|       BL getcode
65|       POP {R1}
66|       MOV R5, #1
67|       ADD R12,R12,#1
```

I get the input string code from the user, check if it satisfies all the requirements, then store it into the #secretcode array.

At this point, R12 is currently used to store my maximum number of guesses, while R5 would be used to store the current guess number (I added 1 to R12 in line 67, since a BLT method would be used later to check if the user has reached the maximum guess number or not in the latter stage).

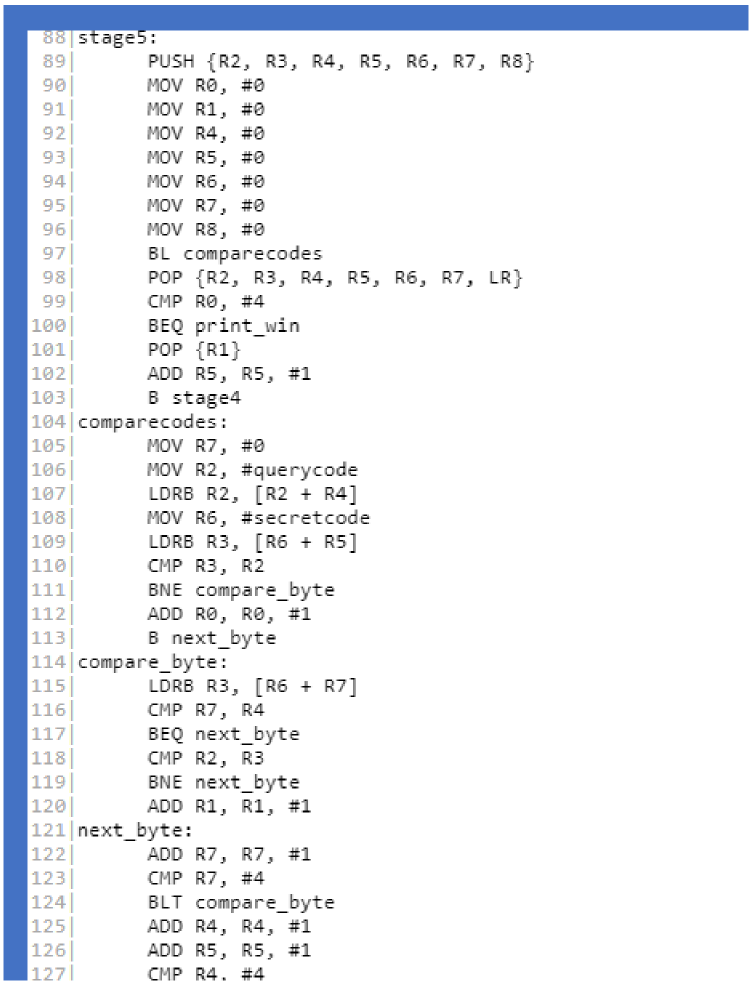
2.4. Stage 4



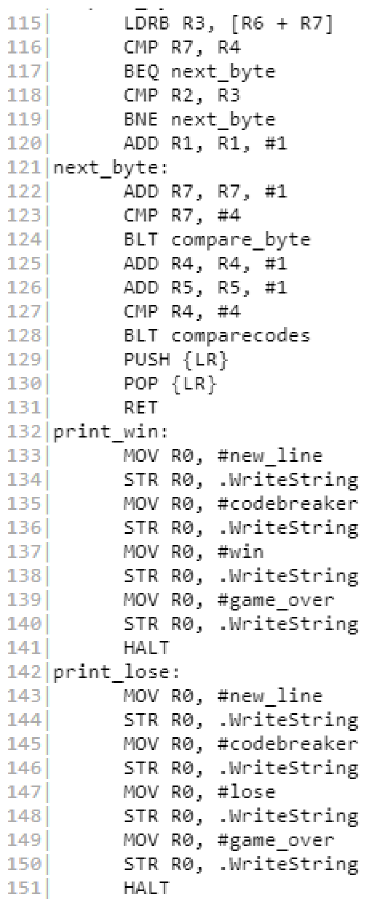
```
stage4:
       PUSH {R2}
       CMP R5, R12
       BLT get_query_code
       POP {R2}
       BL print_lose
;
get_query_code:
       MOV R0, #new_line
       STR R0, .WriteString
       MOV R0, #new_line
       STR R0, .WriteString
       MOV R0, #codebreaker
       STR R0, .WriteString
       MOV R0, #guess_count_message
       STR R0, .WriteString
       STR R5, .WriteSignedNum
       PUSH {R1}
       MOV R1, #querycode
       BL getcode
```

In this stage, I will keep asking for the input of the user until the maximum number of guesses has been reached. This is possible due to the (cmp R5, R12/ BLT get_query_code) functions I used in line 70 and 71. The user input will be stored into the #querycode array to be compared later.

2.5. Stage 5



```
stage5:
      PUSH {R2, R3, R4, R5, R6, R7, R8}
      MOV R0, #0
      MOV R1, #0
      MOV R4, #0
      MOV R5, #0
      MOV R6, #0
      MOV R7, #0
      MOV R8, #0
      BL comparecodes
      POP {R2, R3, R4, R5, R6, R7, LR}
      CMP R0, #4
      BEQ print_win
      POP {R1}
      ADD R5, R5, #1
      B stage4
comparecodes:
      MOV R7, #0
      MOV R2, #querycode
      LDRB R2, [R2 + R4]
      MOV R6, #secretcode
      LDRB R3, [R6 + R5]
      CMP R3, R2
      BNE compare_byte
      ADD R0, R0, #1
      B next_byte
compare_byte:
      LDRB R3, [R6 + R7]
      CMP R7, R4
      BEQ next_byte
      CMP R2, R3
      BNE next_byte
      ADD R1, R1, #1
next_byte:
      ADD R7, R7, #1
      CMP R7, #4
      BLT compare_byte
      ADD R4, R4, #1
      ADD R5, R5, #1
      CMP R4, #4
```

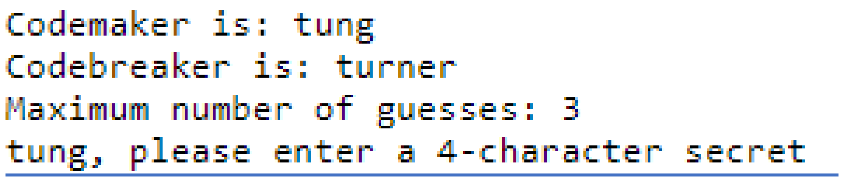


```
        LDRB R3, [R6 + R7]
        CMP R7, R4
        BEQ next_byte
        CMP R2, R3
        BNE next_byte
        ADD R1, R1, #1
next_byte:
        ADD R7, R7, #1
        CMP R7, #4
        BLT compare_byte
        ADD R4, R4, #1
        ADD R5, R5, #1
        CMP R4, #4
        BLT comparecodes
        PUSH {LR}
        POP {LR}
        RET
print_win:
        MOV R0, #new_line
        STR R0, .WriteString
        MOV R0, #codebreaker
        STR R0, .WriteString
        MOV R0, #win
        STR R0, .WriteString
        MOV R0, #game_over
        STR R0, .WriteString
        HALT
print_lose:
        MOV R0, #new_line
        STR R0, .WriteString
        MOV R0, #codebreaker
        STR R0, .WriteString
        MOV R0, #lose
        STR R0, .WriteString
        MOV R0, #game_over
        STR R0, .WriteString
        HALT
```

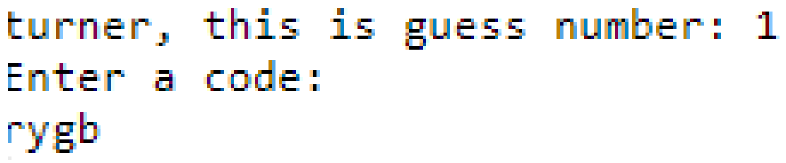
The comparecodes function is used to compare the position of the each character between the querycode and the secretcode array. The compare_bytes function is used when the position do not match: The program will then loop through the other components in the querycode array to find any matching bytes (or matching colors).

After comparing, the #win message will be printed out if the user got the secretcode before reaching the guess limit (matching positions equals to 4), and the #lose message will be printed out if the user is not able to do the above (by using print_win and print_lose methods respectively).

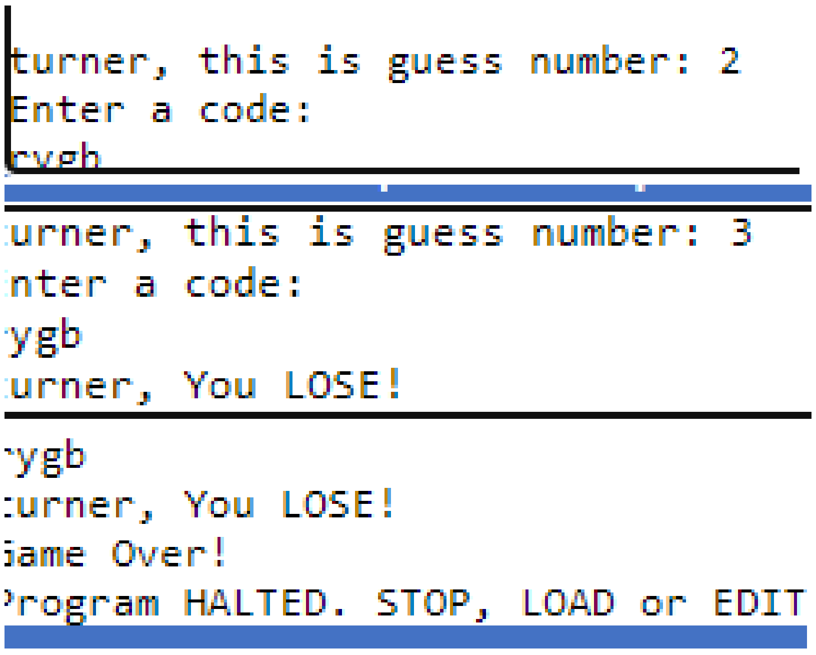
Sample screenshots:



```
Codemaker is: tung
Codebreaker is: turner
Maximum number of guesses: 3
tung, please enter a 4-character secret
```



```
turner, this is guess number: 1
Enter a code:
rygb
```



```
turner, this is guess number: 2
Enter a code:
rygb
```

```
urner, this is guess number: 3
nter a code:
ygb
urner, You LOSE!
```

```
ygb
urner, You LOSE!
ame Over!
rogram HALTED. STOP, LOAD or EDIT
```

3. Unresolved problems

I have only finished the assignment from stage 1 to stage 5a. I still have not completed stage 5b and stage 6 of the assignment (which has graphic display of the pegs).

4. Conclusion

In conclusion, I believe that this assignment 2 is a great way of showing the application of ARM Assembly programming.