Swinburne University of Technology

Computer Technology (COS10031)

Assignment 3 Report

ARM Assembly Programming

Lecturer: Mr. Chandana Deshapriya

Student Name: M.A. Nidula Sanketh Mallikarachchi

Course: UniLink Diploma in IT

Swinburne ID: 104756611

NCHS ID: 2023040050

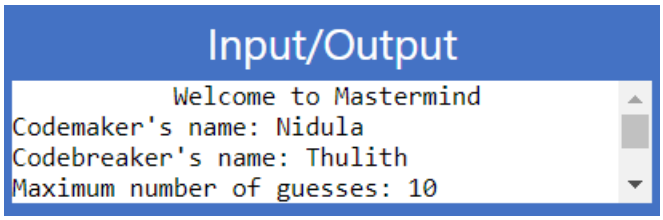
Email: 104756611@student.swin.edu.au

Contact Number: 0741966164

Stage 1 – Game Setup

In stage 1, the register R1 prints string data messages, R12 stores numeric input, and six messages are stored as variables. These messages are stored in the bottom of each stage. Codemaker and codebreaker names are stored in block arrays.

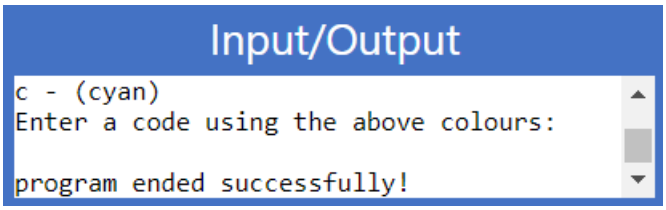
- MOV R1, #msg(n) to load messages to registers.
- STR R1, .WriteString to print msg in input/output terminal
- STR R1, .ReadString to get names from players
- LDR R12, .InputNum to get guess number input
- STR R12, .WriteSignedNum to print number in input/output terminal



Stage 2 – A code entry function

In stage 2, we implemented the get_code: function to filter user input and check its validity. The first validation is through the "too_many_letters:" loop, which checks if more than 4 letters have been added and compares it with 0. If the value is not 0, it indicates an extra character has been added, causing it to be redirected to a function called invalid through. The "invalid" function asks the user to input a valid code, displaying an error message and redirecting them to get_code: until a valid input is provided.
The less code checks if the user inputs less than 4, running a loop four times. Each time, a different letter is loaded to R2, and if 0 is passed, the program displays an error message.

```
LDRB R2, [R0+R1]  //Load a byte from the address in R0+R1 into R2
CMP R2, #65       //Compare R2 with ASCII 'A' (65)
BLT lowercase
CMP R2, #90       //Compare R2 with ASCII 'Z' (90)
BGT lowercase
ADD R2, R2, #32   //Convert uppercase letter to lowercase.
```



The "capital_simple:" function converts uppercase to lowercase letters in user inputs, loading one character from stringData into R2 and checking if ASCII value is within the range of 65 to 90. If the loaded character is within this range, it converts it to lowercase and store back at same location in the array.

```
STRB R2, [R0 + R1]
```

We implemented validation to prevent users from repeating characters by using functions "pick_letter1:" and "pick_letter2:". These functions load a character from the array, compare it with R2, increment R3 values, and exit the loop if all comparisons pass successfully.

We implemented validation to ensure correct input of characters (r, g, b, y, p, c) and load user-input data into R2, comparing it with ASCII values for each character.

- LDRB R2, [R0+R5]
- CMP R2,#114
- BEQ rgbypc2
- CMP R2,#103
- BEQ rgbypc2
- CMP R2,#98
- BEQ rgbypc2
- CMP R2,#121
- BEQ rgbypc2
- CMP R2,#112
- BEQ rgbypc2
- CMP R2,#99
- BEQ rgbypc2
- B invalid

Stage 3 - Getting the Secret Code

The "get_code_maker:" function passes the user string via "STR R0,.ReadSecret", allowing the code maker to enter data without leaking code to the codebreaker and preventing characters from being shown in the display window.
"store_to_secretcode_array:" After converting uppercase letters to lowercase letters, function will transfer data from the stringData array to the secretcode array.

The program checks for input validations using the getcode: function, if invalid, the user receives an error message and must enter the data again

Stage 4 – Query code entry

here we implement new function called "get_code_breaker:" which reads input from.ReadString and stores it in stringData array, while the "store_to_querycode_array:" loop transfers data to querycode.
The invalid function is called when the input is invalid, a common issue in both "get_code_maker" and "get_code_breaker" functions, with R9 set to 1 or 0 respectively.
If codebreaker inputs valid input guess number will decremented in guess_counter
The "get_code_breaker" loop will be executed as many times as the guess number allows, taking the guess number value through R12, and the remaining guess number is displayed each time.

- SUB R12, R12, #1 // counting guesses
- CMP R12, #0
- BGT comparecodes
- BEQ comparecodes

Stage 5 – Query code evaluation

Stage 5(a)

Register R0 keeps track of position and color matches between secretcode and querycode array characters, whereas Register R1 keeps track of color matches

The function "comparecodes:" compares secretcode and querycode array characters, with R5 and R6 registers used for indexing and inputs respectively.

```
CMP R5, R6
BEQ position_same
```

The above code will compare the 2 characters from secretcode and querycode of the same position and see if they are equal. If they are "position_same" will be called and R0 (register that tracks position matches) will be incremented. This compare_positions loop will run 4 times for every character. This is done through the register R7:

```
compare_digit1:
LDRB R6, [R10+#0]
LDRB R5, [R8+#1]
//compares first digit of query code with second digit of secret array
CMP R5, R6
BEQ colour1      //if equal, R1 will be incremented
LDRB R5, [R8+#2]
//compares first digit of query code with third digit of secret array
CMP R5, R6
BEQ colour1
LDRB R5, [R8+#3]
//compares first digit of query code with fourth digit of secret array
CMP R5, R6
BEQ colour1
B compare_digit2
```

The program checks if the codemaker's guesses have been used by the codebreaker, then calls "success_rate" to check if R12 equals 0, then proceeds to compare the new code with the secret code.

Stage 5(b)

The code breaker enters a code, validates it, compares it with the secretcode, and decreases the guess count. The "success_rate" checks if all positions are correct the win function is called. And if not, the lose function is called.

```
CMP R0, #4  // check for wins
BEQ win
CMP R12, #0
BNE get_code_breaker
CMP R0, #4  // if all attempts are gone breaker will lose
BNE lose
win:
MOV R2, #msg16
STR R2, .WriteString
MOV R2, #msg18
STR R2, .WriteString
B print
lose:
MOV R2, #msg17
STR R2, .WriteString
MOV R2, #msg18
STR R2, .WriteString
HALT
```

In the "win" function, it will display a winning message and will end the program. In the "lose" function, it will display a losing message and will end the program.

Input/Output

```
Colour matches: 0
you WIN!
---Game Over---
Program HALTED. STOP, LOAD or EDIT
```

```
cpyb
```