Coursework 2: Systems Programming Report

Jiang Zhaorui and Kang Jiahui

School of Mathematical and Computer Sciences, Heriot-Watt University

***HWU ID*: H00391693 and H00391695**

1. Problem Specification

The objective of the coursework is to design and implement a sophisticated, systems-level application using C and ARM Assembler, which will operate on the Raspberry Pi platform alongside peripheral devices.

This application, named Mastermind, is a two-player game involving a codekeeper and a codebreaker. The codekeeper selects a pattern comprising N code pegs and arranges them within a sequence of N slots, a configuration visible solely to the codekeeper. The codebreaker, over successive rounds, endeavors to deduce the concealed sequence. Following each guess, the codekeeper provides feedback indicating the number of pegs correctly positioned and colored, as well as those possessing the correct color but misplaced. The game concludes either upon the codebreaker correctly identifying the code or upon reaching a predetermined limit of rounds.

2. Hardware Platform and Wiring

The software application is intended to operate on a Raspberry Pi 4, complemented by specific peripheral devices including two light-emitting diodes (LEDs), a push button, and the requisite resistors for each component. These devices will be interconnected to the Raspberry Pi 4 through a breadboard, which can be seen in the Figure 1.

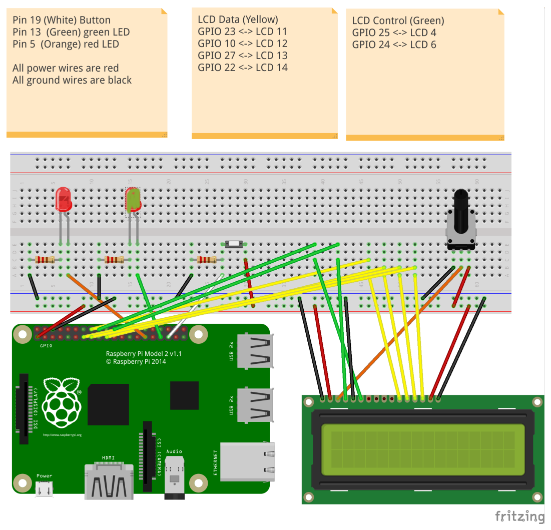


Figure 1. Diagram of Hardware Platform

The detailed wiring of the aforementioned hardware platform about LCD, LED and button can be seen in the following two Tables: Table 1 and Table 2.

Table 1. LCD's GPIO Connections

LCD	GPIO	LCD	GPIO
1	(GND)	9	(unused)
2	(3v Power)	10	(unused)
3	(Potentiometer)	11 (DATA4)	23
4 (RS)	25	12 (DATA5)	10
5 (RW)	(GND)	13 (DATA6)	27
6 (EN)	24	14 (DATA7)	22
7	(unused)	15 (LED+)	(3v Power)
8	(unused)	16 (LED-)	(GND)

Table 2. LCD and Button's Connections

Devices	GPIO
green data LED (right)	13
red control LED (left)	5
button	19

3. Program Logic Structure

3.1. Initialization

- Initialize LED, Button, LCD
- Randomly generate a three-digit secret sequence, with each digit being either 1, 2, or 3
- Display "WELCOME" on the LCD

3.2. Main Loop

Exit after guessing the correct answer and return the number of guesses or loop three times to read a three-digit number input by the player. For each loop, perform the following steps:

Await Button Press and Time Input Completion. Start timing after the first button press, for a total of 4 seconds, during which the input for this cycle is completed. The inputs 1, 2, and 3 correspond to pressing the button 1, 2, or 3 times, respectively. Each input requires at least one press, so start timing after the first press, and end the input after 4 seconds.

Monitor Button Signal for Input Timing. Exit the loop when 4 seconds have passed. If a button press is detected within the loop, increment the count by one. Since the maximum count is 3, the count will not continue to increase after 3 clicks.

Confirm Input with LED Feedback. The red light flashes twice, indicating the end of input. The green light flashes 1 to 3 times, corresponding to the password 1 to 3.

Verify Input and Provide Result. After reading the three inputs, call countMatches to check the relationship between the user input and the secret sequence, and return the exact matches and approximate matches on the LCD.

3.3. Display "SUCCESS" Upon Correct Guess

If the guess is correct, display "SUCCESS" and the number of attempts on the LCD adn then exit the loop.

3.4. End Game if Attempts Exceed 14

If the number of attempts is greater than or equal to 14, break out of the loop and display "game over" on the LCD.

4. Performance Considerations

A discussion of performance-relevant design decisions, and implications on resource consumption regarding 3 aspects from Assembly Integration, Pointer Utilization to Timer Implementation.

4.1. Assembly Integration

Integration of inline assembly instructions within the lcdBinary file to directly interact with hardware components like LEDs and buttons. Use of assembly language enhances program accuracy by enabling direct control over registers, thereby boosting program speed and performance.

4.2. Pointer Utilization

Utilizing pointer arrays to manage both exact and approximate matches throughout program execution. Dynamically allocating memory (malloc) for a pointer array of size 2 simplifies accessing individual match results, streamlining operations through indexing.

4.3. Timer Implementation

Employing a 4-second timer to capture user inputs effectively. The timer kicks off upon the initial button press during number input, facilitating subsequent inputs for improved efficiency and accuracy.

5. Core Functions Description

A list of functions directly accessing the hardware (for LEDs, Button, and LCD. The functions to access the hardware is in the lcdBinary.c file. It has the following structure:

5.1. digitalWrite

Description: This function sends a specified value (LOW or HIGH) to a particular pin.

Implementation: The determination of whether to set the pin to LOW or HIGH. And Inline assembly employed to directly manipulate the GPIO registers, setting or clearing the appropriate bits to achieve the desired output voltage level on the pin.

5.2. pinMode

Description: This function configures the GPIO pin to either input or output mode based on the provided argument.

Implementation: The determination of fSel (function select) and shift bit for the pin is accomplished in C using a switch-case construct. Inline assembly is utilized to set the mode of the pin.

Implementation: C code determines whether to write the value to the set or clear register. Inline assembly modifies the set/clear register to toggle the LED on or off.

5.3. readButton

Description: This function reads the user's button press.

Implementation: C code identifies the gplev0 register based on the pin connected to the button. Inline assembly reads from the determined register and returns the value.

5.4. waitForButton

Description: This function waits until the user presses the button.

Implementation: It relies on the readButton function. No explicit assembler usage here. The functionality is primarily achieved through C code.

6. Matching Function Description

6.1. Input

- **seq1**: pointer to the first integer sequence 1
- **seq2**: pointer to the second integer sequence 2

6.2. Output

- **exact_matches**: stores the number of exact matches
- **approximate_matches**: stores the number of approximate matches

6.3. Match Process

First, obtain the values of exact matches through the exact_loop, marking the positions of exact matches as 0. Then, obtain the values of approximate matches through the main_loop. And the structure of the defined subroutine will be shown as follows:

exact_loop: Iterate through each element of sequence 1 (seq1) and check if it matches the corresponding element in sequence 2 (seq2).

exact_increment: Increment traversal and index variables in the exact_loop.

mark_exact: If seq1[i] matches seq2[i], mark both elements as 0 to prevent interference in the search for approximate matches (approx_matches) in the main_loop.

reset:Reset index and count variables of both sequences (seq1 and seq2) to 0, preparing for a fresh iteration of the main_loop.

main_loop: Iterate through each element (seq1[i]) in sequence 1 and compare it against every element (seq2[j]) in sequence 2 to identify exact matches.

go_next: Advance the traversal and index variables within the main_loop, ensuring the iteration progresses smoothly for comparing elements between the sequences.

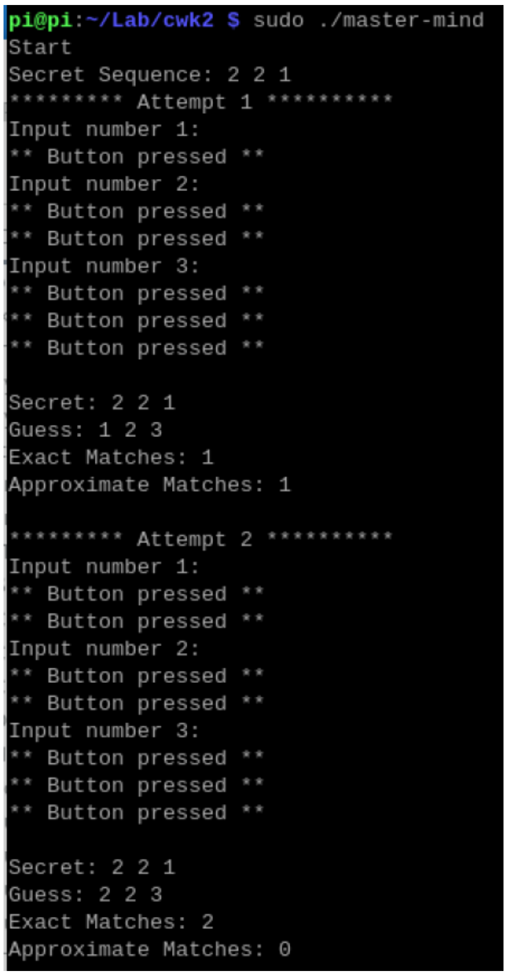
approx_loop: For each element seq1[i], iterate through sequence 2 to check for approximate matches (seq2[j] = seq1[i]).

approx_increment: Increment traversal and index variables in the approx_loop.

mark_approx: If an approximate match is found (seq1[i] = seq2[j]), mark seq2[j] as 0 to avoid duplicate matching with elements of sequence 1.

exit_routine: Once all elements of both sequences have been compared, store the positions of exact and approximate matches in a sequence for further processing.

7. Example Output



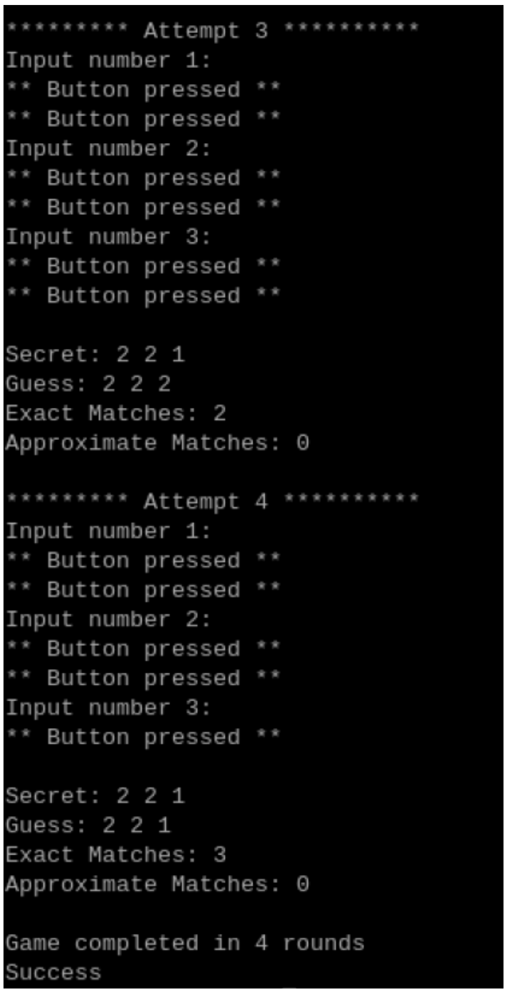
```
pi@pi:~/Lab/cwk2 $ sudo ./master-mind
Start
Secret Sequence: 2 2 1
********* Attempt 1 *********
Input number 1:
** Button pressed **
Input number 2:
** Button pressed **
** Button pressed **
Input number 3:
** Button pressed **
** Button pressed **
** Button pressed **

Secret: 2 2 1
Guess: 1 2 3
Exact Matches: 1
Approximate Matches: 1

********* Attempt 2 *********
Input number 1:
** Button pressed **
** Button pressed **
Input number 2:
** Button pressed **
** Button pressed **
Input number 3:
** Button pressed **
** Button pressed **
** Button pressed **

Secret: 2 2 1
Guess: 2 2 3
Exact Matches: 2
Approximate Matches: 0
```

Figure 2. Terminal Output in Raspberry Pi 4



```
********* Attempt 3 **********
Input number 1:
** Button pressed **
** Button pressed **
Input number 2:
** Button pressed **
** Button pressed **
Input number 3:
** Button pressed **
** Button pressed **

Secret: 2 2 1
Guess: 2 2 2
Exact Matches: 2
Approximate Matches: 0

********* Attempt 4 **********
Input number 1:
** Button pressed **
** Button pressed **
Input number 2:
** Button pressed **
** Button pressed **
Input number 3:
** Button pressed **

Secret: 2 2 1
Guess: 2 2 1
Exact Matches: 3
Approximate Matches: 0

Game completed in 4 rounds
Success
```

Figure 3. Terminal Output in Raspberry Pi 4

8. Summary

In coursework 2, we successfully developed a simple MasterMind game application using C and ARM Assembler on a Raspberry Pi 4. We implemented game logic where the codekeeper generates a random sequence and provides feedback on exact and approximate matches, while allowing user input of sequences via button presses.

Additionally, we supported command-line options for automated testing and debugging, and implemented the matching function in ARM Assembler.

Throughout the process, we gained valuable experience in embedded systems programming, learning about low-level concepts such as GPIO interfacing, memory management, and timing. Although we did not implement a graphical user interface or sound effects, nor optimize the ARM Assembler code for performance beyond meeting the basic requirements, we gained a deep understanding of embedded systems programming.

Overall, this project provided us with valuable experience in embedded systems programming, and there is room for further optimization and additional features in future work.