F28HS Coursework 2

MasterMind

Riffat Khan – rk119
Mohammed Ismael Abidali Shaikh – ms2019

Group: 23

Heriot-Watt University Dubai

March 30, 2022

Contents

Introduction

The aim of this coursework was to develop the classic code-breaking game Mastermind using C and Arm Assembler as the implementation language, with an ability to run on a Raspberry Pi using peripheral devices for user input and output.

The game is a two-player game consisting of a code-keeper and a codebreaker, where the codebreaker must decode the hidden sequence set by the code-keeper. The hidden sequence consists of *N* number of coloured pegs, to be chosen from randomized *C* colours. In each turn, the code-keeper must answer with the correct and approximate guesses i.e., pegs that are of the right colour *and* in the right position or pegs that are of the right colour but *not* in the right position.

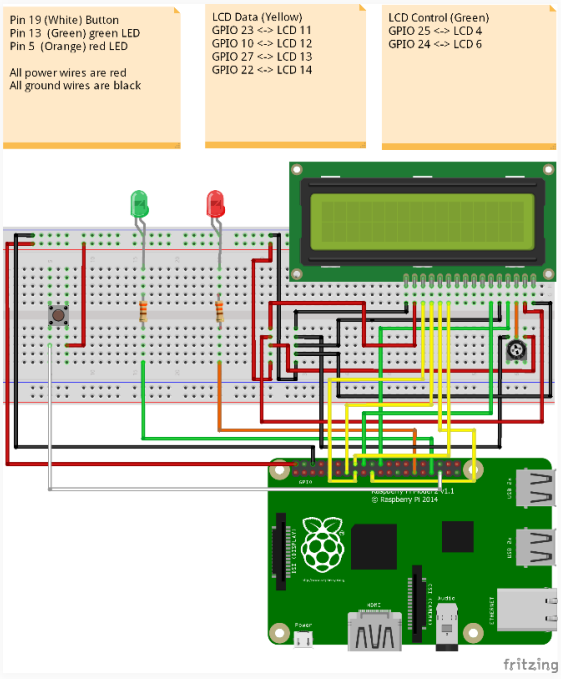
Below is a sample sequence of the game:

```
Secret:  R  G  G
================
Guess1:  B  R  G
Answ1:           1 1
Guess2:  R  B  G
Answ2:           2 0
Guess3:  R  G  G
Answ3:           3 0
Game finished in 3 moves.
```

Hardware Specification

The attached peripherals consist of two LEDS (One red and one green), a button and a 16x2 LCD display with a potentiometer connected to it for controlling the contrast. These are connected to the GPIO pins of the Raspberry Pi through a breadboard.

The wiring for the peripherals is based on the Fritzing diagram provided in the Coursework specifications:



LEDs

Two LEDs are connected to the Raspberry Pi using GPIO Pins for controlling information and data output. Each LED utilizes a resistor in order to control the flow of the current. For turning on or off an LED, the current state of the pin is set directly through Inline Assembly code. Both LEDs are powered directly through the GPIO pins.

GPIO Connections:

1. Red LED - GPIO 5
2. Green LED - GPIO 13

Button

A button for user input is connected to the Raspberry Pi using a GPIO pin for data transfer and a connection to the 5V power supply pin. The button utilizes a resistor for controlling the flow of the current. The reading of the voltage states (High and Low) of the button are done through Inline Assembly.

GPIO Connection:

1. Button - GPIO 19

LCD

A 16x2 LCD display is connected to the Raspberry Pi using GPIO pins and a 5V power supply for LCD *Data* and LCD *Control*. Additionally, a potentiometer is attached to the LCD in order to control the contrast of the LCD Display, which is connected to LCD 3.

GPIO Connections for LCD Power:

1. LCD 2 – 5V Pin
2. LCD 15 – 5V Pin

GPIO Connections for LCD *Data*:

1. LCD 11 – GPIO 23
2. LCD 12 – GPIO 10
3. LCD 13 – GPIO 27
4. LCD 14 – GPIO 22

GPIO Connections for LCD *Control*:

1. LCD 4 – GPIO 25
2. LCD 6 – GPIO 24

Code Structure

The game was coded in a combination of C and Inline Assembly.

The working of the game is implemented in C, which is primarily contained in file *master-mind.c*.

Functions requiring direct hardware-level access to LEDs, Button and the LCD are implemented using Inline Assembly in file *lcdbinary.c*.

In addition to direct hardware-level access, the matching function for matching input sequences to secret sequences was implemented using Inline Assembly in file *mm-matches.c*.

C

The functions required for the main functioning of the game are as follows:

1. initSeq
   This function is used for initializing a randomized sequence consisting of N length and C values (repeatable), which must be decoded by the codebreaker.
   For initializing a randomized sequence, numbers between 1 and 3 (inclusive) are randomly inserted into an int array using srand.
2. countMatches
   This function is where the matching of the secret sequence and guessed sequence takes place.
   A temporary array is created for flagging colors which have already been *seen/iterated*. Following this, the user input is iterated through for matching correct colors in the same position, and these are flagged.

Similarly, a nested for loop is used for iterating for approx. values, where if colors which are correct, but in the incorrect position along with not being flagged are counted. Exact and approximate values are then encoded into one value and returned.

3. showMatches
   This function is used for displayed the exact and approximate matches on the terminal based on user.
4. readSeq
   This function is used for reading a 3-digit value from the user and inserting it into an array. The array will then be used for comparing to the secret sequence in a different.
5. Main
   The main function is where the game is initiated from. Several modes are defined in the main method to be used as arguments for execution. These are defined below in the *Usage* section.
   Variables and temporary arrays are defined for array

Inline Assembly

1. CountMatches function
   The count matches function is implemented in inline assembly unlike the other functions. The assembly code corresponds exactly to the C code which was initially implemented. Branches and subroutines were used in a clear manner. The function counts how many entries in the secret sequence match the entries in guess sequence as well calculate the number approximate matches i.e. number of occurrences in a different position but same color (number). It returns the exact and approximate matches, both encoded in one value, using the concat helper function. To make the function a bit more efficient, when the exact values in the right positions have been guess, the calculation for approximate is skipped to save time and speed up the process.

Usage

The command line interface is as follows:

./cw2 [-v] [-d] [-u <sequence1> <sequence2>] [-s <secret sequence>]

Where -v, -d, -u & -s are modes that can be passed to the command line.

The program can be executed with 5 different modes:

1. Empty/No mode
   Running the executable file without including a mode in the command line results in the execution of the mastermind game
2. Verbose
   The Verbose mode executed as -v returns extended information about the program. Below are a few examples of how the mode works.
   -v with debug mode ON:

```
pi@raspberrypi:~/Documents/Tutorial/CW2Code $ sudo ./cw2 -v -d
Settings for running the program
Verbose is ON
Debug is ON
Unittest is OFF
```

-v with unit test mode ON and debug OFF:

```
pi@raspberrypi:~/Documents/Tutorial/CW2Code $ sudo ./cw2 -v -u 123 321
1st argument = 123
2nd argument = 321
Settings for running the program
Verbose is ON
Debug is OFF
Unittest is ON
Testing match function with sequences 123 and 321
1 exact
2 approximate
```

3. Debug
   The Debug mode executed as -d displays the randomly generated secret sequence so that the user can check the exact and approximate matches.
4. Unit Test
   The Unit Test mode executed as -u <seq1> <seq2> automatically tests the exact and approximate matches between two sequences passed as arguments.
5. Secret Sequence
   The Secret Sequence mode executed as -s <secret> allows the user to enter the secret sequence and act as the code-keeper. A sequence for the secret must be passed as an argument.

Some modes such as -v, -d & -u can be used together.

Hardware-Level Access

• A list of functions directly accessing the hardware (for LEDs, Button, and LCD display) and which parts of the function use assembler and which use C

Inline Assembly Hardware

For direct hardware-level access, the following functions coded in Inline Assembly were used:

1. digitalWrite
   This function sets the state of pin for the LEDs and LCD.
2. pinMode
   This function is mainly used for setting the mode of Pins.
3. writeLED
   This function is used for setting the state of LEDs. It's quite similar to digitalWrite, except for the fact that this function is solely for LEDs.
4. readButton
   This function returns the current state of the button, for which the pin number of the button must be passed as a parameter.

Tests

The following tests were conducted on the program:

1. Unit Testing Setup (from main method using -u mode) as specified in coursework specs

```
pi@raspberrypi:~/Documents/Tutorial/CW2Code $ sudo ./cw2 -u 121 313
0 exact
1 approximate
```

2. Test script using test.sh

```
pi@raspberrypi:~/Documents/Tutorial/CW2Code $ sh ./test.sh
Cmd: ./cw2 -u 123 321
Output:
1 exact
2 approximate
Expected:
1 exact
2 approximate
.. OK
Cmd: ./cw2 -u 121 313
Output:
0 exact
1 approximate
Expected:
0 exact
1 approximate
.. OK
Cmd: ./cw2 -u 132 321
Output:
0 exact
3 approximate
Expected:
0 exact
3 approximate
.. OK
```

... (tests 4 to 9)

```
Cmd: ./cw2 -u 312 312
Output:
3 exact
0 approximate
Expected:
3 exact
0 approximate
.. OK
10 of 10 tests are OK
```

3. Using testm.c

```
pi@raspberrypi:~/Documents/Tutorial/CW2Code $ ./testm
Running tests of matches function with 10 pairs of random input sequences ...
Matches (encoded) (in C):   11
Matches (encoded) (in Asm): 11
1 exact
1 approximate
1 exact
1 approximate
__ result OK
Matches (encoded) (in C):   11
Matches (encoded) (in Asm): 11
1 exact
1 approximate
1 exact
1 approximate
__ result OK
```

… (tests 3 to 9)

```
1 exact
2 approximate
1 exact
2 approximate
__ result OK
10 out of 10 tests OK
```

Sample Execution

A sample execution of the program in debug mode i.e., where the secret is displayed beforehand; is shown below:

```
pi@raspberrypi:~/Documents/Tutorial/CW2Code $ sudo ./cw2 -d

Secret : B R G
Raspberry Pi LCD driver, for a 16x2 display (4-bit wiring)
Printing welcome message on the LCD display ...

Press ENTER to continue:

======================

Guess 1: BRR
2 exact
0 approximate

Guess 2: BGR
1 exact
2 approximate

Guess 3: RBG
1 exact
2 approximate

Guess 4: BRO
2 exact
0 approximate

Guess 5: BRG
3 exact
0 approximate

Congratulations! You broke the code in 5 attempts!
```

The working of the LEDs and a test run through the entire program on the Raspberry Pi is shown in the attached video.

F28HS_CW2Video.mp4

Summary

A completely working implementation of the game using C and Assembly was achieved. We gained a lot of hands-on experience with the Raspberry Pi and got to know more about direct hardware access.

Our count matches function was implemented in Inline Assembly rather than ARM Assembly. This allowed us to adapt more towards inline assembly and understand more about direct hardware-level interaction.

We also managed to implement a 16x2 LCD Display despite being a group of 2 members.

Concluding, we believe the coursework enhanced our critical thinking skills as well as our adaptability.