F28HS-CW2 Report

Lucca Anthony Marcondes Browning – H00369673

Ian Jossic – H00375186

Heriot-Watt University, Edinburgh

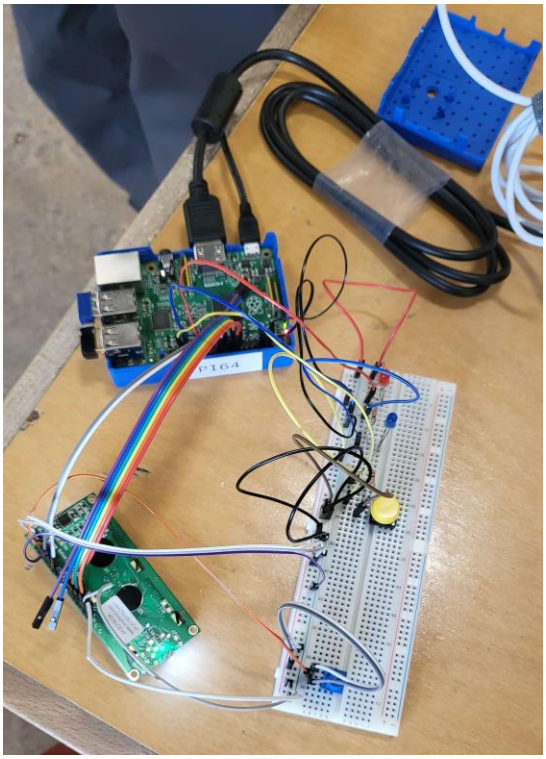
Problem Specification:

This coursework specifies an incomplete set of programs, in both C and ARM Assembler that are supposed to complete the function of a game, whose input is interacted with by the user pressing a button on a breadboard connected to a raspberry pi a certain number of times to signify the sequence of numbers of their guess. The output consists of printing the exact matches and the approximate matches onto an LED display. If the input is correct, a green LED blinks 3 times and a red one turns on. If the output is incorrect, nothing happens. If the user runs out of guesses, the red LED blinks 3 times to signify the game is over.

The solution to completing this game would be to gather a small team to split the tasks that need to be undertaken, whether it's writing C code, writing ARM assembly code, assembling a machine that fits the purpose outlined by the code (which would be the Raspberry Pi in this case), or combining and testing the ARM and Assembler code on the machine.

Quite early on, Jossic and Browning decided to split the work so that the workload of the two would remain relatively even. This would lead to the problem being in the most efficient amount of time possible. Browning would predominantly write the ARM assembler code and some of the C code as well as assemble the Raspberry Pi and write the bulk of the report, whereas Jossic would write most of the C code, compile and test all the ARM and C code on the Raspberry Pi, and write parts of the report. The goal to be achieved was to create a working version of the Mastermind game.

Hardware Specification and Wiring:



The primary piece of hardware being used in this project was a Raspberry Pi Generation 2 model wired to a breadboard that is connected to 2 LEDs through GPIO, a green one and a red one. The breadboard also connects to a button and an LED display to the Raspberry Pi. As previously mentioned, the input device in this Raspberry Pi in the specification of this project is a button on a breadboard that is pressed a certain number of times to signify the sequence of numbers of their guess. The output consists of the LED display, whose purpose is to print the exact matches and the approximate matches of any given guess. The output also consists of a green LED and a red LED. If the input matches the expected value, a green LED blinks 3 times and a red one turns on. If the input does not match the expected value is, nothing happens. If the user runs out of guesses (6 in this case), the red LED blinks 3 times to signify the game is over.

Figure 1: The Wired Raspberry Pi

Code Structure:

Primarily, the code consists of two programs, the mm-matches.s program, which checks if a sequence of button inputs is an exact match, approximate match, or not a match at all to an expected value. It does this using ARM assembler code. This 'matching function' is called inside of the master-mind.c program, which has the first 990 or so lines of code creating helper functions, which are then used inside of the main method.

The main method waits for the first button press to start the game, and counts that button press as the user's first input. Then, it continues waiting for more button presses as inputs to the user's 'guess'. Once the input is entered, the program compares the input code to the secret/expected code. If the user guesses the right code, the green LED blinks 3 times (while the red LED is on), and once the game is over (either the user guesses 6 times or a win is achieved), the red LED blinks 3 times.

Performance Relevant Design Decisions:

Throughout the creation of the mm-matches.s and master-mind.c programs, it was a primary focus of Browning and Jossic to lower the complexity of the code.

For example, inside of the mm-matches.s program, Browning used the fewest number of CPU registers possible by reusing registers with pre-existing, nonfunctional data in them, leading to a decrease in the overall memory being used in the program. This can be seen in the 69th line of code in mm-matches.s, where the code "LDR R1, [R0], #4" assigns a new value to the register R1. However, R1 was being used previously in lines 48-55 to compare guess values with expected values in a loop. This technique can also be seen in line 74 where the code "LDR R3, [R1] assigns a new value to register R3. R3 was also used previously in lines 48-55. It should also be noted that Browning avoided nesting loops in the Assembly code, leading to a worst-case complexity of O(n) when it could have been O(n^2) if the program was written more poorly.

Inside of the master-mind.c program, Jossic avoided increasing complexity by using as few loops as possible in the main method as well as using a lot of memory allocation in the helper functions to avoid memory leaks and remove items from memory when they are not needed. For example, it is used in the initSeq() function, which initializes the secret/expected sequence. It's also used in the main function, which sets the sizes for a few variables (seq1, seq2, cpy1 and cpy2) and the size of the sequence theSeq.

List of Functions that Access Hardware:

- Section: Hardware Interface
    - digitalWrite()
        - This function uses C for its if-else statement.
        - It uses an inline assembler to send a value (LOW/HIGH) on pin number.
    - pinMode()

        - This function uses C for its if-else statement.
        - It uses an inline assembler to set the mode of a GPIO pin to input/output.
    - writeLED()
        - A C function that calls digitalWrite(), and therefore accesses hardware
    - readButton()
        - This function uses C entirely to read data from the GPIO.
    - waitForButton()
        - This function uses C entirely to read data from the GPIO.
- Section: LCD Functions.
    - These functions all use C entirely to read and write data from the hardware.
        - lcdPutCommand()
        - lcdPut4Command()
        - lcdHome()
        - lcdClear()
        - lcdPositon()
        - lcdDisplay()
        - lcdCursor()
        - lcdCursorBlink()
        - lcdPutChar()
        - lcdPuts()

Interface Discussion on mm-master.s:

The interface consists of two very important, but rather simple parts. Firstly, is the input, which consists of two parameters passed to it from the master-mind.c program. One of which is the secret/expected code, which is put into the second register (R2), and the other one is the guessed code, which is put into the third register (R3). The output from mm-master.s's matches subroutine is a single integer that encodes the counts of exact and approximate matches. The exact matches are represented by the tens place digit, and the approximate matches are represented by the ones place digit. For example, if there are 2 exact matches and 1 approximate match, the integer returned in R0 would be 21.

Sample execution of the program in Debug Mode:

Browning and Jossic wrote the code but were unable to test it due to a recurring error which was not able to be traced or fixed. The error consists of:

/tmp/*filename*.s: Assembler messages:

/tmp/*filename*.s:118: Error: bad instruction `oris r3,r2,r1'

/tmp/*filename*.s:173 Error: bad instruction `oris r3,r2,r1'

Summary:

As it stands, all that was achieved was the completion of the mm-matches.s program, which is fully functional. However, Browning and Jossic were only able to get very small parts of the master-mind-c program to work independently of the rest of the program because of the two errors above that persisted for the entire project, no matter what they tried, whenever they tried to run the master-mind.c program. The report was done to the best of their abilities given the limitations of the project that was run. Next time, Browning and Jossic will attempt to consult a wider variety of sources in order to fix future problems with their code to produce a program that runs more optimally.