COS10004 Computer Systems

Assignment 2 Overview

This assignment requires knowledge of ARM assembly programming with the ARMlite simulator and assumes you have covered all modules up to and including Week 10.

Purpose: Implement a simplified version of the board game Mastermind in ARM assembly for the ARMlite simulator

Task: Follow the defined sequence of stages to implement the functionality for the board game Mastermind (described in more detail below). Each stage should be thoroughly tested and is expected to work as described in each stage when loaded into the ARMlite simulator.

Due: 11:59pm Tuesday November 1st

Assessment: The assignment is worth 20% of your assessment for this unit.

Submission Instructions:
Your completed submission must be made through Canvas - (Go to Assignment 2 under "Assignments" before the due date/time. Everyday day late will incur a 10% deduction.

Each submission should be ***zip file containing***:

- All .ASM files for each stage completed
- a report (Word doc or PDF) containing:
    - Your name, student number, unit code and lab session
    - An overall outline of your solution, with a brief description of each function you have defined.
    - Any assumptions you have made
    - Any unresolved problems with your program
    - Pasted screenshots of providing evidence of your working program for each stage completed (one shot per stage is fine).

Resources:

- All lectures and slides from Week 7-Week 10 on Canvas
- ARMlite programming reference manual:
- Richard Pawson's "Computer Science from the Metal Up"
- ASCII character code lookup table
- Mastermind introduction
- Tutor consultation times

Academic Integrity

This assignment is not a group assignment. While high level discussions with your peers and with your tutor are fine and expected, the work submitted must be your own, and will be subject to similarity testing. You should not share your solution with anyone, or make available online. Any encountered cases of plagiarism will immediately result in a zero grade being awarded for the assignment for all parties involved, and probable further disciplinary action taken. If you are struggling with the assignment, you should make use of the numerous hours of tutor consultation available to you (as advertised on Canvas and above under resources).

Mastermind

Background

Mastermind is a code-breaking game for two players. One player becomes the *codemaker*, the other the *codebreaker*. The codemaker chooses a pattern of four code pegs, each being one of 6 colours. While in general, duplicates are allowed, in our version of the game, no duplicates or blanks are allowed. The codemaker places the chosen pattern in the four holes covered by the shield, visible to the codemaker but not to the codebreaker.

The codebreaker tries to guess the pattern, in both order and colour, within some number of turns (typically between 8 to 12). Each guess is made by placing a row of code pegs on the decoding board. Once placed, the codemaker provides feedback by placing from zero to four key pegs in the small holes of the row with the guess. A coloured or black key peg is placed for each code peg from the guess which is correct in both colour and position. A white key peg indicates the existence of a correct colour code peg but placed in the wrong position.

Once feedback is provided, another guess is made; guesses and feedback continue to alternate until either the codebreaker guesses correctly (in which case the codebreaker wins), or all rows on the decoding board are full (i.e.., the number of allowed turns is reached,) in which case the codemaker wins).

Here is a video explaining the game as well: https://www.youtube.com/watch?v=dMHxyulGrEk

The Assignment

You have been tasked with writing a simplified version of Mastermind in ARM assembly, for execution on the ARMlite Simulator. Your version will not be concerned with the visual feedback elements of the game as described above, but rather the basic logic and game play, using text entry and text display (except for Bonus Stage 6!).

The assignment is divided into 5 stages (and a bonus marks Stage 6), guiding you through different functionality that is needed. Each stage is worth a percentage of the total mark. You need only complete up to Stage 5 to get full marks. You must work through each stage in sequence and save your solution for each stage in an ASM file named after that stage. You will submit all files at the end, along with a design document describing your solution (design document described above).

To get full marks for each stage, it must work as described and will be tested in the ARMlite simulator by one of the teaching team. Partial marks are available for each stage; however emphasis will be on working solutions. A rubric will be published on Canvas to guide the allocation of marks for each stage). To fulfil all requirements of the assignment, you will have to have completed all modules up to and including Week 10.

In addition to the implemented functionality, your solution will also be marked according to its clarity and proper use of conventions, in particular Application Binary Interface conventions governing function definitions (see Week 10 lectures).

Stage 1 over page

Stage 1 – Game Setup (30% of marks)

Before the game can start, some setup options need to be set. Both players, the codemaker and the codebreaker, should be able to enter their name. Also, the game should allow the user to set the number of allowable guesses for the codebreaker to crack the code.

To do all this, write an ARM assembly program that:

- asks for the codemaker's name, reads it in and stores it in a character array labelled codemaker
- asks for the codebreaker's name, reads it in, and stores it in a character array labelled codebreaker:
- asks for the maximum number of queries allowed by the codebreaker to guess the code, and stores the number in a dedicated register
- Once all the above is read in, the program should print the following message, each on separate lines:

  `Codebreaker is` *<insert codebreaker name>*

  `Codemaker is` *<insert codemaker name>,*

  `Maximum number of guesses:` *<insert number of guesses>*

You should replace the above "<…>" parts with the actual data entered.

Save your solution using the filename stage1.asm

Stage 2 over page

Stage 2 – A code entry function (25% of marks)

Both the codemaker and codebreaker will need to be able to enter a four-character code, with each character being one of six colours:

'r' – red

'g' – green

'b' – blue

'y' – yellow

'p' – purple

'c' - cyan

Codes will be entered as 4-character length strings using a single letter indicator for each colour as shown above.

For simplicity, we will assume no duplicates of any colours in the set code, or the query code.

So for example, a code of yellow-purple-blue-red pegs would be entered as "ypbr", or a code of green-red-yellow-purple" would be entered as "gryp".

Because this functionality is needed by both the codemaker and codebreaker, it makes sense to write this as a function, so we can call it every time a code entry is needed from a user.

To do all this, add a function called "`getcode`:" to your Stage 1 program that does the following:

- Prints to the text display, the message: "`Enter a code:` "
- Reads in a 4-character string from the user as per the format described above and stores it in an array. The array to store the string in should be passed in as a parameter to the function
- Tests the input string for the following:
    - That each character entered is only one of 'r', 'g', 'b', 'y' , 'p', or 'c'. For this you will need to use an ASCII character code lookup table to find the codes for each lowercase character.
    - That the total number of characters entered is exactly 4 (no more, no less).
    - If either of the above is not satisfied, the function should repeat all the above until the user enters a valid string

To demonstrate it works, add a call to this function in your main program.

Save your full solution to a file called stage2.asm.

Stage 3 over page

Stage 3 - Getting the Secret Code (15% of marks)

The next thing we need to add is functionality to take in a secret code from the codemaker. In the previous stage you wrote a function that does most of the work.

Modify your Stage 2 program so that, after doing the initial setup, the codemaker is able to enter a string of 4 characters indicating a colour selection for each peg. For full marks, the program should:

- Define a character array `secretcode`: to hold the secret code entered
- display a message on the console saying "<codemaker name>, please enter a 4-character secret code" (where <codemaker name> is replaced by the actual codemaker's name
- uses the function getcode: defined in Stage 2 to read in a code and store it in the array secretcode.

Save your solution using the filename stage3.asm

Stage 4 over page

Stage 4 – Query code entry (10% of marks)

We now need to modify the program to allow the codebreaker to enter query codes. We won't worry about comparing each query code with the secret code yet, but we will first make sure we can repeatedly read in query codes up to the maximum number of allowed guess.

Modify your Stage 3 program so that after the codebreaker has entered the secret code, the program:

- Defines an array `querycode:` to hold the current query code entered
- Prints to the text display the message:
  "<codebreaker name>, this is guess number: <N>",
  where <codebreaker name> is the name entered in Stage 1, and N is the current number of guesses left.
- implements a loop that repeatedly asks and accepts a 4-peg query code from the codebreaker
- stores the entered query code in an array called `querycode`:
- terminates the loop when the number of iterations reaches the nominated number of allowable turns (entered as input in Stage 1)

Save your solution using the filename stage4.asm

Stage 5 over page

Stage 5 – Query code evaluation (20% of marks)

Each time the codebreaker enters a query code, the code needs to be compared with the secret code and evaluated for correctness. For each peg in the query code, one of three cases may occur:

- **Case 1**: the query peg is a direct match with its corresponding peg in the secret code, OR
- **Case 2**: the query peg is a match with at least one other peg in the secret code but not in the correct position, OR
- **Case 3**: the peg has no match in the secret code

So for example, if the secret code was "rgpy", and the query code was "rpgb", then the counts would be: Case 1 = 1, and Case 2 = 2. Or if the query code was "ypgr", then the counts would be Case 1 = 0, and Case 2 = 4. And of course, an exact match with the secret code would generate the count Case 1 = 4, Case 2 = 0.

As stated earlier, for simplicity you can assume there will be no duplicate colours in either code.

Stage 5(a) (15% of marks)

Based on the above, add a function called "`comparecodes`:" to your program that compares the query code with the secret code according to the above rules. Specifically, the function should:

- accept both string codes as input parameters
- count how many query pegs are in the correct position (Case 1),
- count how many pegs are not in the same position but do have a colour match with at least one other peg in the secret code (Case 2).
- Both values should be passed back from the function as return values using R0 (Case 1 count), and R1 (Case 2 count).

Save your solution using the filename stage5a.asm

Stage 5(b) (5% of marks)

Using the output of `comparecodes`**:** defined in Stage 5(a), modify your program to give feedback to the codebreaker. Specifically, after each query code is compared with the secret code, your program should print the following message to screen:

"`Position matches: <X>, Colour matches: <Y>`"

where X is the Case 1 count, and Y is the Case 2 count. In addition:

- if the number of exact matches (Case 1) is equal to 4, then the program should print:

  "`<codebreaker name>, you WIN!`"

- If the maximum number of guesses is reached without the code being guessed, then the program should print:

  "`<codebreaker name>, you LOSE!`"

- And in either of the above cases, the program then terminates with the message:

  "`Game Over!`"

Save your solution using the filename stage5b.asm .

This completes the assignment. See over page for optional bonus Stage 6.

Bonus Stage 6 - Visualising guess history (up to 10% of bonus marks available)

You do not need to do this to get full marks!

It is useful for the codebreaker to see all their previous guesses. For this, we can use the graphic display in *Medium-resolution mode*. Modify your program so that for every query code entered by the codebreaker, a 4-pixel horizontal line is drawn where each pixel represents the colour in the corresponding query code. For example, if the codebreaker enters "rpgb", then a 4-pixel line of pixel colours red, red, green and blue should be drawn. When the Codebreaker enters their next query code, the next line drawn should be in the row directly below the previous.

Other graphic elements to more closely resemble the actual visuals of the board game (see video link above) will also be considered worthy of bonus marks. You should first discuss this with your tutor though.

You should write this as a function that takes the query code array and the current guess number as parameters, and based on this, draws the 4-pixel line in the correct location. Your function should conform with Application Binary Interface conventions

Save your solution using the filename stage6.asm

Note that up to 10 bonus marks are available for this stage, awardable up to but not exceeding the total 100 marks available for the assignment. Bonus marks are only available for fully working solutions to this stage.