CS 0447 Computer Organization and Assembly Language

Iterm Project

Introduction

In this project, y ting a simplified version of wordle game using MIPS assembly.

The principle is the same as the original one. But we'll introduce a simplification: Players can try to guess non-existing words, like `lsdkf`. Otherwise, it's pretty much the same but in text.

Start early

The deadline will approach fast! Life happens, sickness happens, so if you start early, you can minimize the impact. Do a little bit every day! 1 hour every day! 30 minutes every day! SOMETHING!

How the game will work

The program starts by printing a hello message introducing itself and presents a menu to select what to do next. You don't need to print the same thing! Do your own prints!


```
==================================
Welcome to my little game, let's play
==================================
What do you want to do?
    (1) Play
    (2) Quit
```

At this point, if the user selects an invalid value, you can just print the menu again and ask again. If the user selects the option to quit, the program should terminate calling syscall 10. If the user selects the option to play, the game begins.

Example 1 - Lose a game

```
Make your guess: hello
 h (e)(l)[l] o
Make your guess: spell
 s  p (e)[l](l)
Make your guess: pixel
 p (i) x (e)(l)
```

```
Make your guess: water
 w (a) t (e) r
Make your guess: world
 w  o  r [l] d
Whoops, it seems you could not guess :(
 The word was:
```

Example 2 - Win

```
Make your gues
 p (i) x (e)(l
Make your gues
 h (e)(l)[l] o
Make your guess:
 s  w [i] n [e]
Make your guess: guile
(g) u [i][l][e]
Make your guess: agile
[a][g][i][l][e]
Yay :)
 The word was indeed: agile
```

Example 3 - Multiple games

```
==========
Welcome to Wordle, let's play a game
==========
What do you want to do?
    (1) Play
    (2) Quit
1

Make your guess: agile
(a) g  i  l  e
Make your guess: skimp
[s] k  i (m) p
Make your guess: smart
[s][m][a][r][t]
Yay :)
 The word was indeed: smart

==========
Welcome to Wordle, let's play a game
==========
What do you want to do?
    (1) Play
    (2) Quit
1

Make your guess: smart
[s][m][a][r][t]
```

```
Yay :)
 The word was indeed: smart

====================================
Welcome to Wordle, let's play a game
====================================
What do you wa
    (1) Play
    (2) Quit
1

Make your gues
[s] m  a  r  t
Make your guess: spell
[s][p][e][l][l]
Yay :)
 The word was indeed: spell

====================================
Welcome to Wordle, let's play a game
====================================
What do you want to do?
    (1) Play
    (2) Quit
2
```


Explanation

As you can see, during the game you will ask the player to make a guess. Read a 5-letter word from the keyboard. If your program allows the user to input a word larger than 5 letters, validate the word and ask another time. There is a syscall that limits the number of chars that can be input, check it if you'd like to use it instead.

When a letter is not in the word, I just print it on its own **x** - I added spaces around it for consistency. When a letter was found on the word, but in the wrong location, I print it with parentheses **(x)**. And if a letter is in the right location, I print it with square brackets **[x]**.

The player only has 5 guesses.

Your assignment

Plan

This includes data structures you are planning to use, user inputs that may be invalid and you need to account for, etc. 1. Think of which functions you will need to implement, and what they will do: 1) Start from the main function and split your program into multiple steps. 2) This plan is not going to be enforced, but it should

be thought through. 2. Think of possible user inputs, and how they will impact the program negatively. Run over some examples with pen and paper and see/make sure the behaviour is correct.

Implement

Implement the M that executes the game described above!

1. It begins ome message and an explanation of how to proceed.
2. If you sel its.
3. If you sel egins.
4. Then your program will:
    1. Randomly select a word from a list of at least 5!
    2. 5 times
    3. Ask the player for a 5-letter word.
    4. Wait for user input.
        - If the input is a 5-letter string, continue.
        - Otherwise, ask again.
    5. Print the user input using the symbols `[]` `()` to indicate the correctness.
    6. If the player guessed the word, print a victory message, and stop playing.
    7. If you ran out of tries, print a losing message.
5. When the game round is over, display the menu again.

The welcome message

I try to restrict your imagination as little as possible in this project. You do your own thing, as long as it fits the project! So, use the welcome message to explain to the user anything that is not clear in the game.

User input

Some syscalls that you may want to use:

- `print character.`
- `read character.`
- `read string.`
- `random int range.`

Make sure your string variables (the ones you use for user input) can hold enough space for a user input (including **`\0`**)!

Think about which function you need

If you are using strings, you may want to implement functions to compare/copy/etc those strings. (Maybe not! There are many ways to implement this code). Think about those thin

The game is com eps. Print and handle menu, run game loop, ask and handle word etc.

Think about thos unctions you need.

Storing data

How will you store your list of words? Maybe an array of strings? How can you do that? What sort of limitations will it have and how will you access each string? (maybe a function?).

Because all strings are the same size, I used a matrix of words!

Debug mode

You must have a variable named **debug** at the top of your code. The grader must be able to change its value to **1** to enable debug mode, or to **0** to disable debug mode!

In debug mode, the game should print the random word once it's selected! It will help you and the grader figuring out if all is working.

Project Stages

In order to help you be aware of your progress, I will recommend a series of mile markers to help you divide up the work. You can, of course, ignore these if you wish. However, if you find you need some direction, by all means follow along.

I've divided this into three stages. You could consider accomplishing each stage one by one.

Stage 1 - Plan! Create the data structures you think you'll need and print

Solving a project is more than writing code! So, your first task should be designing the solution.

- How are you doing user interaction?
- What information do you need to store?
- What will execution look like?
- What are the parts of your program and how do they interact?

So, for the first stage:

- Read the project description and understand it.
- Implement the strings and other variables you think you need. (Like messages for the player)
- Write the ons (e.g., I wrote a function to ask for a 5-letter word, on
- Break the er pieces (each should be a funcion).

If you finish ear ge 2.

Stage 2 - Parsing

In stage 2, I suggest you tackle parsing user input. Write a function that takes user input and decides if each input letter is: (1) in the correct position, (2) in the word but in the wrong position, or (3) not in the word at all. You can print the feedback to the user as you go (making the code simpler).

At this point, you are really close to the end!

If you finish early, move on to stage 3.

Stage 3 - Finish the project

You should have a few weeks to the deadline, now it's time for testing!

Helpful Tidbits

Starting the code

This is a very simple program in a higher-level language! But it is much more complex in assembly. As such, here are some advice for developing your program.

Plan ahead and start by writing high level comments on how you plan to approach the problem:

- If you are not sure what to write, ask for help.
- If you need, then write the program in a high-level language, draw a diagram, write pseudo-code, and then translate that into MISP assembly.

Testing

DO NOT TRY TO WRITE THE WHOLE PROGRAM BEFORE TESTING IT!!!!

- Really! Do not do it! It's the easiest way to get overwhelmed and confused without knowing what to do!
- Implement small parts of the code and test them!

Split your code into functions

Use functions! They will help you manage the cognitive load. Here is a starting point!

```
main:
    jal print_

_main_loop:
    ...
  jal   get_in
  <stuff>
    j   _main_loop
    <moar stuff>
```

Submission

Submit a single ZIP file with your project named studentID_MidtermProj.zip (e.g., 2022141520000_ MidtermProj.zip). In the zip file, there should be NO folder, just the following files:

- Your wordle.asm file. (Put your name and student ID at the top of the file in comments!)
- A readme.txt file (DO NOT SUBMIT A README.DOCX/README.PDF. SUBMIT A PLAIN TEXT FILE. PLEASE.) which should contain: a) your name, b) your student ID, c) anything that does not work, d) anything else you think might help the grader grade your project more easily.

Submit into the Blackboard. Let me know immediately if there are any problems submitting your work.