**1. A project report (in Word format) covering**

**a) a description of the program**

1. MIPS assembly Lexathon is a is a 9-letter word puzzle game where you discover as many words as possible that contain the central letter.
2. It's a vocabulary building game that is fun and suitable for all users with any level of English proficiency and for any age.
3. Test your skills; there is always a 9-letter word to find among the scrambled letters!
4. Every word you find gets you more time and increases your score!
5. The faster you react and the more words you find from the English dictionary.
6. Once you’ve found as many words as you can, enter '0'(zero) to ‘Give up’ to maximize your score
7. Scores are awarded per the number of correct words and the player receives 1 point for every letter.

**b) the challenges that you had and how did you overcome them**

- Some of the challenges I had implementing algorithms to perform some tasks. To overcome this, I wrote them in C++ code and translated into MIPS assembly.

**c) what you have learned by doing the project**

By doing the project, I got to learn about the MIPS architecture, understand how data is represented by a computer and understand how a computer manipulates data and perform computations.

It helped me learn how to write more efficient high level code when I understand what the compiler is going.

Also, I got some experience with developing a game.

**d) a discussion about algorithms and techniques used in the program**

- Initialize total word count

- Initialize score

- Initialize time remaining to 60 seconds

- reset positon to 0

- gets next word and stores it.

- aligns to next word

- Pick random 9 letter word and its acronyms and display

- if end of current word go check against user input

- else store the current char in the next position in increment.

- if character fail word is not equal-fail

- increment position

- increment user input position

- compare right word

- if correct, go next word input

- if wrong, input another word

- if end check score

- display result

- loop

**e) contributions of each team member,**

I had initially implemented a separate complete Lexathon game while the other two team members (James Fritz and Connor McCourt) worked on a separate game.

We then merged the functions and procedures and used below parts of my game.

1. Timer functions.

2. User interface.

3. User interface background functions

**f) any suggestions you may have (optional)**

**2. A short video clip demonstrating the program in action**

The video clip demonstrating the program in action can be found using below link.

<https://youtu.be/v0ient_Dm9o>

**3. All code that are needed to run your program**

Attached below files (CS3340\_Project.zip) needed to run the game.

* LexathonGame.asm
* ScoringFunctions.asm
* TimerFunctions.asm
* WordListFunctions.asm
* StringFunctions.asm

**4. A user manual on how to run and how to use the program**

**The features of this implementation of Lexathon include:**

1. A main menu that prompts the player to enter '0' for instructions on how to play or '1' to start game (0 or 1).
2. Provides the player will a grid containing 9 letters (3x3 grid) from which to form as many words as possible.
3. An input reader that allows the user any input, including one that causes game exit, and the user input will be checked for whether it is correct length or invalid, includes center character generated in the grid, and is a valid word in a dictionary. Also, gives the player the option to end the game at any time by inputting a '0'.
4. A timer that counts down initially from 60, and adds 20 seconds for every correct possible word entered by the user, ending the game at 0 seconds, and displaying the time with each submission.
5. A scorekeeping module that is printed at the end for timer that is calculated from words per minute and words found.

**Limitations:**

1. The entirety of the "GUI" will be in the console in MARS.
2. The timer update is only shown on console with every submission attempt/at game over.
3. Some grid generations will only yield a single digit number of possible solutions arbitrarily.
4. A small selection of common words does not appear in this dictionary, as not every variation of a root word is covered in it. EX: If "dancing" is there, "danced" might not be.
5. Repeating the central letter many times in a string input may crash the program.
6. Entering multiple 0s counts as one 0 in the context of the program and ends the game accordingly.

**How to start the game**

1. Download all contents of this project into a folder
2. Start "Mars4\_5.jar"
3. Build and run
4. Have fun playing!