Name: T Abishek Entry Number: 2019CS10407
Name: Gaurav Jain Entry Number: 2019CS10349

COL216: Assignment 3

Introduction:

This assignment aims to build an interpreter/simulator to handle a subset of MIPS assembly language instructions by maintaining data structures of registers and memory and executing each instruction line by line, and simultaneously showing the effect of these instructions by printing the registers values. The method employed is building a data structure of an array of registers along with an array of bytes to handle memory and building associated functions that work on these data structures to read and change values in these data structures. We read the input file and store these instructions in the memory by using integers to represent different instructions and then execute them when the file is read while also taking care of errors in the input file and then subsequently print the results as seen in the registers.

Approach to the code:

The user is supposed to provide the input file in a .txt file which should contain the MIPS assembly code that is supposed to be executed. The code should satisfy the following criteria:

1) The simulator has 32 registers, zero, sp and r1 to r30
2) add, sub, mul, beq, bne, slt, j, lw, sw, addi are the only supported instructions
3) Labels aren't supported. The code should directly give the offset address integer whenever branch or jump is requested.
4) In lw and sw instruction, there must be a space between offset and the register. Ex: 320 ($r1)
5) Comments haven't been handled. Writing comments would give an error.

The code reads the instructions line by line, storing it as the instruction along with its parameters after parsing for easy execution later on and then stores the pointers to these instructions in memory data structure. After parsing, execution of the code begins. It starts from the first instruction and proceeds accordingly. While executing, errors in the input are checked. The correctness of the instruction and its parameters is checked before executing them. If any error is found, a relevant error message is printed and the program exits. Each instruction is checked against the different possibilities, and once it is matched, the relevant code is executed where the parameters are checked before calling the relevant functions defined earlier on the memory and register data structures to modify and read the data in them. After each instruction is executed, a function is called which prints all the register's current data line by line for checking the current state of the registers. The pc variable which keeps track of which instruction is being executed is also updated accordingly depending on the return variable of the 'process' function and is also checked if it goes out of bounds before updating the pc variable. This helps us to implement the jump and branch function when required. A map is also maintained to keep track of the no. of times each instruction is being executed, which is updated accordingly whenever the relevant instruction is being executed. After all the instructions are executed, the statistics are also printed using the map that was maintained earlier, and the no. of clock cycles are also printed by summing the map values.

Name: T Abishek Entry Number: 2019CS10407
Name: Gaurav Jain Entry Number: 2019CS10349

Our testing strategy involved thinking and figuring out what can happen when a user interacts with our program and then implementing the code to handle such cases. Cases like entering invalid characters, entering more arguments in the expression than the operators require, entering wrong operators, entering wrong arguments have all been handled while writing the program. The testing strategy also involved trying out the different scenarios using various test cases that test all the different scenarios that can transpire while running the program. The different kinds of test cases that were used are included along with the document.