COL216 ASSIGNMENT 3

Avantika Agarwal (2019CS10338) Aryan Jain (2019CS10334)

APPROACH AND DESIGN CHOICES

We are required to make a compiler for MIPS instructions. The compiler is expected to load and execute **add, sub, mul, beq, bne, slt, j, lw, sw, addi** instructions, while also handling and properly processing any labels and/or comments in the file. The program is further required to maintain a structure representing a memory of size 2^20 as well as 32 integer registers (including the stack pointer).

As our design choices, we made the following decisions:

1. The memory is byte addressable
2. The memory is an array of integers initialized to 0
3. The instructions are stored in a vector, and the portion of memory array corresponding to the instructions, is inaccessible for data storage
4. The 32 integer registers are also initialized to 0 (decimal value), except the stack pointer, which points at the last address of the memory array. The values are converted to hexadecimal values while printing the register values to console.
5. lw and sw instructions can be called both by using memory address or by using offset values
6. j, bne, beq instructions take a label as their third argument
7. Each instruction ends with a new line (even the last line of instructions)

We took the following approach for the implementation:

1. Input is read line by line, and all the instructions are separated into tokens which are stored in an instruction vector as tuples of 4 strings (command and arguments, in case arguments are less than 3, rest of the strings are empty). An error message is shown if the text file has some syntax error.
2. When reading labels in the text file, they are stored in a hashmap, with keys as the labels, and values as the instruction memory address where the label is stored. This helps to locate the required address when branching occurs.
3. The variable instructions_end_index stores the address from where data memory starts.
4. The program counter initialized to 0 starts executing the first instruction in the memory, and after execution of each instruction, either the program counter is incremented by 4 bytes, or is set to point at the required instruction memory address in case of branching.
5. The execution of the program stops when the program counter starts pointing at instructions_end_index.
6. The number of clock cycles is incremented by 1 after each instruction has been executed, and the number of times the corresponding instruction has been executed (this is maintained in an array indexed by the instruction address) is also increased by 1. The integer registers are also printed to the console after execution of each operation.
7. For implementing the individual instructions, we first check whether the input register values are valid or not. If they are valid, the first token of the instruction (corresponding to the

operation to be performed) is read, based on which the corresponding implementing function is called.

8. The operation is performed, and the result is stored in an integer register or data memory as needed.

ERROR HANDLING

The code prints an error message in the following situations:

1. If the instruction is not among the ones specified at the beginning
2. If the input file has a syntax error
3. If the load word and store word instructions try to load from or store to instruction memory addresses
4. If the user tries to access a register which is not accessible to user, or if a non-existent register is specified in the input file
5. If the integer range overflows, C++ throws an error
6. If a label has been defined twice, or if branching instruction has non-existing label as parameter

Once an error message has been printed, the execution of the program stops.

TESTING STRATEGY

We have considered the following categories of testcases:

1. General tests: test1.txt, test3.txt, test4.txt, test5.txt
2. Invalid data memory address: test2.txt, test9.txt
3. Invalid instruction: test6.txt
4. Invalid register argument (trying to access inaccessible register): test7.txt
5. Infinite loop: test8.txt
6. Syntax error: test10.txt
7. Non-existent register: test11.txt
8. Invalid labels: test12.txt, test13.txt