COL216 Assignment3

Aniket Gupta - 2019CS10327
Ishaan Singh - 2019CS10359

12 March 2021

1 Approach

The main aim of the assignment is to create an interpreter for a subset of MIPS assembly language. The instructions that are handled are add, sub, mul, beq, bne, slt, j, lw, sw and addi. The overall code is very modular with different functions performing different actions and different data-structures representing the different types of system memory (register memory, memory for storing instructions, memory for data-storage etc.). The overall execution can be divided in two parts:

- Reading, recognising and storing instructions in the memory with proper error handling wherever required.
- Performing the operations given in the instructions and giving output after executing each instruction.

1.1 Reading Instructions

Each of the instructions will be stored in the form of a struct with different fields. Single line of a MIPS programme can contain only single instruction. We read the instructions from the input file one line at a time in string format. Then the string is split into the exact instruction type (i.e. add, sub, mul etc.), the related register values and integer values associated with the instruction based on the space and comma delimiters. Each instruction is stored in an array of instructs. During reading the input, syntax errors are detected but are raised only after the entire input file is read.

1.2 Performing Operations

Once all the instructions are read, the execution step starts. Global pointer PC indicates the current instruction to be performed in the instructions array. To perform the instructions, global pointer PC is maintained that indicates the instruction number that is to be performed. Different types of instructions are performed by calling different functions. Based on the type of instruction

pointed by PC, proper function is called. These functions make the changes required in the register memory and the data storage memory. After performing the instruction, PC pointer is updated. In case of beq, bne and j instructions, the PC pointer is updated to the next instruction no. pointed by the label. In other operations, it is increased by one.

1.3 Output

Once all the instructions are executed, the output is printed on the terminal. Output shows the status of registers after each instruction. Apart form this, it prints the no. of times each instruction is performed and the total no. of cycles taken to complete the execution (assuming each instruction takes single clock cycle).

1.4 Error Handling

It takes care of different types of errors that can occur.

- In case of a syntax error detected during reading the instructions from the input file, it raises the error after the complete input file is read.
- In case of input that causes infinite loop during execution, warning is raised and nothing else is printed on the terminal.

2 Testing Strategy

For testing, we tried to create an exhaustive set of types of testcases possible. Most of the valid testcases will belong to atleast one of these types. The types of testcases covered are as follows:

- Single Instruction only
- Multiple instructions in the input file and each instruction is performed one time
- Multiple instructions in the input file and each instruction is performed one or more times
- Multiple instructions in the input file, but some are not used due to jump or branch instruction
- Syntactical Errors (comma left, invalid operation type, multiple instructions in a line, integer in place of register)
- Infinite loop formed due to jump or branch instruction
- Negative label in jump or branch statement

We have added a file containing testcases "TestcasesA3.txt" and the corresponding outputs in "outputs.txt".