COL 216: Computer Architecture

Assignment 3

Preface:

In this assignment we have a designed an interpreter which takes a text (.txt) file as input which contains a subset of mips instruction and executes it using some internal data structures. We have handled **add, sub, mul, beq, bne, slt, j, lw, sw, addi.**

After each instruction we have printed relevant statistics like clock cycles and number of instructions in addition to the values of the 32 int registers.

Our Approach:

In this assignment we have defined a new class named **MIPS** through which we are implementing all the facilities. In addition to this we have also designed our own implementation of a **stack** using **linked list** for implementing the stack functionality of the mips.

In the MIPS class we have implemented the following things:

- An **instruction map** which stores the instructions token with their line number as the key. The token is a vector of string.
- A **memory space** of 10^20 bytes using array.
- A **map** of the 32 registers is stored with name of the registers (string) as key and the value being the content of the register.
- A **clock** is maintained.
- A **Stack** is maintained named S for implementing the stack pointer of the mips.

Other than this various helper functions are implemented to maintain these structures. We have a function named **executeInst** which takes the instruction map and the executes the instructions by type-matching the tokens.

Input Handling:

We take a text file as input line by line and then parse a particular line into tokens. This is done by the function named **lineToken.** This returns us a vector of tokens after separating out comments and whitespace etc. When we get this vector of tokens, we insert it into out instruction map which line no being the key. This process is done till all the line of the input file is tokenized and mapped in the instruction.

After the tokenization we call the **executeInst** function provided in the mips class which executes the instruction set line by line and updates relevant structures.

Constraints:

1. Initially all the registered are set to 0.
2. Memory values used will be multiple of 4 only, no intermediate memory change available.
3. No direct integer addition using “add”, you have to provide the register values whose corresponding values are to be added, well, “addi” is specifically for that purpose only use it!!
4. Since all the instructions provided were integer operations, float registered are not handled.
5. Maximum memory size is 2^20, any value more than that will cause “Overflow Error”.

Error Handling:

For every command we have type matched them the line tokens with the format that it is supposed to be and if there is some type mismatch then we ended the program with appropriate error line. Also if type is correct but the execution is not possible or meaningful like jumping to a line out of scope of program or reading or writing the value in memory that is more than the range, then also we have thrown appropriate error.

We have tried our code to be as complete as possible, that can handle every possible combination of syntactically correct or incorrect commands.

Moreover, we have also taken into account the ignoring of comments.

Testing Strategy:

After successfully writing the code, for testing we choose the method of induction on number of instructions to prove the correctness of the program.

Base Case:

For the base we first executed the code for every possible command and checked the output with hand computation. All the outputs were correct implying that our code works fine for every command give.

Induction Hypothesis:

Now let the code works correctly for k number of instructions.

Induction Step:

Now for k+1 instruction, by Base case our code works correct for last line and by induction hypothesis it works correct for other k+1 lines, hence the code works like charm:).

For thorough testing we used the pieces of MIPS code from the previous assignments containing the instructions that are given in question, then executed the code on them.

NOTE: All the test cases are available on file “Testcases.txt