COL216 Assignment-3

Devanshi Khatsuriya 2019CS10344 | Ishita Chaudhary 2019CS10360

C++ program that reads a MIPS assembly language program as input and interprets it instructions by maintaining internal data structures representing processor components such as Register File and Memory

Design Choices:

The following data types and functions are used in C++ code:

- *memory*: an array to represent memory and store instructions and data. Its size is 2^{18} (2^{20} bytes, as given in the problem statement, with 4 bytes per instruction) is declared with the data type *inst_data.* This data type is of the form (Boolean, string, integer, integer, integer, string, Boolean, integer, integer).
  Boolean: this is true if the keyword is an instruction and false if it's a data.
  String: this stores the label name. If there is no label, this is an empty string.
  Boolean: this is false if the corresponding operation involves an integer and true if it involves a register.
  This array represents memory stores the instructions and data
- *r:* an array to store 32 registers. It has $r0 at index 0 to $r30 at index 30. It also has $zero at last, i.e., index 31.
- An unordered map *label_map* is created to maintains (label name, index) pair.
- An integer PC maintains the index of the memory array, which has to be executed next.

Other design choices:

- An error function is made to report different errors including trying to overwrite a zero register, encountering an undeclared label name and invalid access to memory.
- We have implemented memory as a block of 4 bytes each. So, an offset of 1 in indirect addressing would mean 1 block in our memory, i.e., 4 bytes.

- An error is thrown if the instructions attempt to read or write at locations with index exceeding 2^{18} (maximum size) in memory array.
- A similar error is raised when an attempt to read or write (via sw or lw) at locations in memory where instructions are stored.
- A syntax error is reported at necessary places, along with the line number and an exception is thrown.
- A separate function is made to report unexpected errors. Such as, when a keyword is encountered which is neither an instruction nor a label.

Implementation and C++ Code Explanation:

The test file should be in the same directory as main.cpp, and its name should be given in command line (incorrect name throws an error "File not found"). This text file is read line by line, line number is initialised to zero in the beginning.

A function *parse_line* takes istringstream object by reference, and reads all lines of the text file. An empty line and comments encountered are ignored. The first word is checked and jumps to its corresponding code block (add, sub, mul, beq, bne, slt, j, lw, sw, addi).

The function *extract_arguments* is called which separates the arguments on commas and whitespaces (and removes comments, if any). The extra white spaces and tabs on either ends of the string/argument are trimmed by function *trim.* The function *read_register_value returns* the index where the register must be stored in array "r".

If the instruction is add, sub, mul, bne, beq or slt: a Boolean *aux* (true if register, false if an integer) checks if the operation is with an immediate value or a register. Label in these cases is an empty string.

If the instruction is addi, everything is similar as above, except the need to check *aux*, as it's always an integer.

If the instruction is j, the function *read_label_value* is called, which checks if the word is valid (has letters, numbers and underscores only) and returns it.

If the instruction is lw and sw, *aux* is true when it is indirect addressing mode using parentheses and false means integer address in memory is given.

In case the line doesn't has an instruction, it is checked for the presence of a label in it (by checking the presence of ":"). If the conditions are satisfied, the label is inserted into *label_map* along with corresponding index.

In each case, the memory at corresponding index is updated along with an increment in index.

When the program reaches the end of file, a run function is executed. While PC is less than the index of the last occupied item of array memory, a function *run_instruction* is executed. It performs the function corresponding to the instruction name and if the operation is with a register or an integer (in case of add, mul etc.) and updates the value of registers.

In case of "sw", the memory at the required address is updated.

After completion, PC, the count of instruction and cycle number are incremented. The value of all 32 registers is printed in hexadecimal code along with cycle number.

If the instruction was "j", the *label_map* is used to find the index of the corresponding label and instead of incrementing the PC it is set to the index found. Similarly, in case of "bne" and "beq", if there is a label involved then PC is updated in the above fashion, else it is simply incremented.

Whereas in case of "lw", only the register value is updated with the corresponding value of memory item.

Finally, after exiting from the while loop, when the PC becomes equal to the last index of memory array. The final value of all registers in hexadecimal along with total number of clock cycles and total no. of times each instruction was executed is printed.

Syntax errors, execution errors and any other type of error is handled and reported at each step, if the required conditions are not satisfied.

Testing Strategy:

To run the C++ file, run the following commands on your terminal (ubuntu machine used).

$ g++ main.cpp -o output -std=c++11

$ ./output TEST_FILE_NAME.txt

1. Test Case 0: Empty file. All register values, total number of clock cycles and number of instructions executed are zero.
2. Test Case 1: This file has add, sub, defined label, comments, slt with register comparison and bne. It also has tabs instead of spaces. The program executes with correct outputs at each cycle and accurate final results.
3. Test Case 2: This file has addi, lw, sw with indirect addressing mode using parenthesis. The program executes with correct outputs at each cycle and accurate final results.
4. Test Case 3: This file has add function with the same register 10 times, also the addition is with immediate value. The program executes with correct outputs at each cycle and accurate final results.
5. Test Case 4: This has an instruction: j branch, where "branch" is not defined anywhere in the text file. The program throws an error with message, "undeclared label name".
6. Test Case 5: This file has addi with immediate value (add different indentation), mul with registers with a comment concatenated at the end of instruction. The program executes with correct outputs at each cycle and accurate final results.
7. Test Case 6: This has an instruction: addi $zero, $r0,5. Program throws an error with message," cannot write to $zero register".
8. Test Case 7: This has an instruction: add $r0, 3. Program throws syntax error.
9. Test Case 8: This has an instruction: lw $r0. Program throws syntax error.