Design

Yohandi (120040025)

Overview

This project is about executing a translated machine obtained from a assembly language code, that is MIPS. As a matter of fact, the computer that we all know can not run the MIPS code directly. The code requires itself to be translated to a machine code containing 0-bit and 1-bit. After being translated, the computer will be able to read the machine code and execute the operation directly from the given task.

There are actually 55 MIPS instructions that are to be transliterated. The instructions are categorized into three types those are R-type, I-type, and J-type. Each type represents a different interpretation of the machine code. Each of the instructions falls into one category. Because of this reason, each of the instructions must be solved case by case. The thoughts and ideas will be explained later in the Processing Logic part.

Data Type and Data Structure

1. Boolean

In C++, boolean variables are defined by a true or false value. It is possible to use the integer type of variable to indicate 0 as a false value and 1 as a true value. However, the boolean variable saves up a lot of the allocated memories.

In this program, the boolean variable is used for many cases that only consider two possible choices. In its example, it is used to determine whether the current state is in the `.text` state or `.data` state.

2. Integer

In C++, an integer variable is a data type that stores numerical values. As explained in the boolean part, the integer variable can be used to store values such as 0, 1, 109, and possibly many numbers within some specific ranges. In its example, it is used to determine the current number line while doing the calculation of a label address.

Aside from the basic integer variable, int32_t is also used in this program. Mainly because the variable provides similar advantages as the normal integer type with 32 bits. This type is specifically used in storing the information of a label address. Another type used in this project is int64_t.

3. Array

In C++, an array is a sequential value list. The array allows storing, changing, and accessing a specific element based on the index parameter in constant time. An array is used to obtain the given arguments in this program while running the executable program.

4. Bitset

Bitset is a container in C++ STL (Standard Template Library). Mainly, the use of bitsets creates simplicity in handling some variables that primarily use a bit. For example, while converting a label address from the int32_t type to binary, it can be easily done with this provided bitset from the STL.

5. String

A string is a collection of variable characters, which usually is used to store text. The string variable can also be considered as a data structure provided in C++ STL. This variable is a core in this program since all of the inputs and outputs use this string type. For example, each of the instructions can be saved to a string such as `add $t3 $t1 $t2`. Moreover, the code's core address needs to be stored inside a string variable for its label.

6. Map

In C++, creating an array-like type that has a custom key is not an easy task. That is why having a map data structure that supports any type of key is a must. Although using this map data structure costs more storage data by log multiplication in terms of space complexity, it eases the implementation job. In its example, a map variable is used to store the label as its key and its address as its value. In order to get the address, the value can be simply returned by calling the map directly. Moreover, the map is also used as a constant library to store opcode in I-type and J-type instructions, function code in R-type instructions, and the binary information of register numbers.

7. Vector

A vector, in its simplest way, is an array that is dynamic. The size of the allocated memory can be added and removed like a stack data structure bases. In this program, one of the uses of a vector is to store some strings compacted in one variable. Another thing, the vector that contains those strings can be stored again inside a vector, making it another dynamic array in a two-dimensional type.

Processing Logic

First, it must be noticed that the goal of this project is to execute every instructions found in the list of machine codes obtained from the translation of a MIPS code. The MIPS code itself contains some sections such as `.data` and `.text`. The instructions that are mentioned before are mainly taken from the `.text` section. Because of this reason, the work of separating the static contents will be done by the phase1 class. The contents are the static data section, the text address pointer save, and the data address pointer save. The label address and data address will be stored in the labelTable class as a custom data structure. Subsequently, phase2 class works as the executer of the cleaned version of instructions. Lastly, the main source of this project will read the input file, write a new output file.

The class of phase1 is intentionally designed to deal with the both `.data` and `.text` section for assembling. After receiving the input file name from the main, the function `separated_text_type` and `separated_data_type` will first determine the location of the section; after confirming it to be on their respective sections, the function will begin to consider each of the lines input given. This work can be handled by maintaining a boolean expression that will be set true once a substring of `.data` or `.text` is found in the line. Not to be left behind, the task of removing comments must also be worked on. This can be simply done by scanning each of the characters in the line, and once a character `#` is found, the current character and the after characters will be ignored. As a result, each line in the `.data` section can be concluded in either five types of data: `.ascii`, `.asciiz`, `.word`, `.byte`, and `.half`. As another result, each line in the `.text` section can be concluded that it contains either label declaration, instruction, or both. In the `.data` section, every static data found will be saved according to the pointer in their address. In the `.text` section, for the line that contains some labels, the addresses of those labels need to be saved in the labelTable class, which later will be explained on how it works. For the line that contains an instruction will be ignored since the machine code is already provided by the tester.

The class of phase2 is designed for the execution of each of the machine code that is translated MIPS instructions. Phase2 class mainly executes the instruction case by case. The file contains several dictionaries to keep the constant available without having to reconsider each key value. Those dictionaries are stored in the map data structure. The function `execute` in this file is called exactly as many as the instructions are called. In the `execute` function, it first determines the operation given in the instruction and solves it case by case. Besides, other functions aside from the main `execute` function are also called in this function with the purpose of simplicity. Bitset, which is provided by C++ Standard Template Library, comes in handy for almost every operation that are required to be done. It allows a conversion of any integer type of value into a binary string. The conversion can be done with `bitset<number_of_bits>(value)` property by bitset.

The class of labelTable is intentionally created as storage. The storage saves the address of all collected data and labels in a map data structure. The file contains four functions where two of the functions are to save both of the label and data address, and the other two are to access the address value based on the label variable. The calculation of absolute label address is `0x400000 + 4 x line_number`. The calculation of relative label address is the value of absolute address - `0x400000 + 4 x line_number`. The function in labelTable will often be called in phase1 class to save label addresses. The function in labelTable will also often be called in phase2 class to get the label addresses information. The label addresses information will later be used as the work of a loop or even a nested loop.

Output

Running the Project

This project requires both the Qt Creator application and the CMake application to run. From the given .zip, there is a folder containing all required source codes and a CMakeLists.txt file to determine the main compile for execution.

After successfully compiling, another folder with a `build` name as its prefix will occur. Tester is required to employ the some considered `.asm` and `.txt` files as the test cases in the build folder.

Tester is required to specify five additional arguments after `.exe`, additional arguments after that are ignored. Those three arguments are the asm file, the converted asm file, the checkpoint file, the input file, and the output file. For window users, simply find the executable program in the build folder and run it in the command prompt with `simulator.exe test.asm test.txt test_checkpts.txt test.in test.out` command, where the `test.asm` refers to the MIPS code file, the `test.txt` refers to a file that is translated from the MIPS code. The `test_checkpts.txt` refers to a checkpoint that works as a debug helper. Moreover both of the `test.in` and `test.out` work as the both input and output test cases that work as code result. Similar to window users, linux users are also able to run it in the terminal with the same command along with the `./` as its prefix. However, some problems occur for windows users where the executable program launches an error message stating that it requires to be reinstalled. For this problem, please run it from the Qt Creator application directly. First, open the Qt Creator application and choose this project. Then, in the projects tab, find the run tab and fill the command line arguments starting from the input file until the expected output file (note that the name of the executable program should not be stated).