ICS LAB A

Introduction

In this Lab, you'll need to implement a tiny LC3 assembler.

If you choose to use the framework we provide:

- Your task is to replace all `TO BE DONE` with the correct code.
- You'll need to learn how to use `Makefile`.
- We recommend you to finish this Lab in Linux (vlab is a good option).

Otherwise, you can also write the assembler from scratch by yourself.

You may refer to chapter 7.3 `The Assembly Process` in the textbook for help.

Overview

```
.
├── Makefile      // the Makefile to be used for this lab
├── src
|   ├── assembler.cpp
|   ├── assembler.h
|   └── main.cpp
└── test
    ├── expected  // expected outputs for testcases
    |   ├── test1.bin
    |   ├── test2.bin
    |   └── test3.bin
    └── testcases
        ├── test1.asm
        ├── test2.asm
        └── test3.asm
```

You'll need to complete all `TO BE DONE` in `assembler.h` and `assembler.cpp`

Assignment

- The correct code will get you 95% of the marks for this experiment.
- Report accounts for the rest 5%.

You only need to hand in your report renamed `PB21xxxxxx_姓名_labA.pdf`.

（附加实验代码需要检查验收，具体方式：线下/线上待通知）

Makefile

In this lab, we compile our codes with the help of `make`.

type command

```
make
```

will do:

- generate `main.o` and `assembler.o` from `main.cpp` and `assembler.cpp`
- link `main.o` and `assembler.o` into `assembler`

`assembler` is the executable file of our assembler

type command

```
make clean
```

will remove `assembler.o`, `main.o`, and `assembler`

Part1: The first pass

Step1: Format every line

Before we process any line read from a `.asm` file, we need to do some pre-processing:

- remove comments (comments starting with `;`).
- convert the line into uppercase.
- replace all commas with whitespace (for splitting).
- replace all "\t\n\r\f\v" with whitespace (so `TAB` and other control chars become whitespace).
- remove the leading and trailing whitespaces.
    - implement `Trim` function first

To simplify the problem, we assume the operand of `.STRINGZ` only consists of uppercase English letters or numbers.

- You can improve the framework so that it supports `.STRINGZ` with any string operand, but it's not required.

Complete `FormatLine` function in `assembler.h` in this step.

Step2: Store label

In the first pass of assembly, you need to store labels with their addresses in the Symbol Table.

Complete `LineLabelSplit` function in `assembler.cpp` in this step.

This function accepts a formatted line(in step1) and:

- If the first word in the line is not an opcode, then treat it as a label. Store it with its corresponding address in the Symbol Table. Return the line with the label removed.
- Otherwise, simply return the line.

`label_map` is a member of the `assembler` class, you can store the label with its address by:

```
label_map.AddLabel(/* something here */)
```

Step3: Complete the first pass

Complete `firstPass` function in `assembler.cpp` in this step.

You only need to modify `current_address`.

Part2: The second pass

In the second pass:

```cpp
int assembler::secondPass(std::string &output_filename) {
    // Scan #2:
    // Translate
    std::ofstream output_file;
    // Create the output file
    output_file.open(output_filename);
    if (!output_file) {
        // @ Error at output file
        return -20;
    }

    for (const auto &command : commands) {
        const unsigned address = std::get<0>(command);
        const std::string command_content = std::get<1>(command);
        const CommandType command_type = std::get<2>(command);
        auto command_stream = std::stringstream(command_content);

        if (command_type == CommandType::PSEUDO) {
            // Pseudo
            output_file << TranslatePseudo(command_stream) << std::endl;
        } else {
            // LC3 command
            output_file << TranslateCommand(command_stream, address)
                        << std::endl;
        }
    }

    // Close the output file
    output_file.close();
    // OK flag
    return 0;
}
```

The function calls `TranslatePseudo` or `TranslateCommand` to convert the `asm` code into `bin` format.

Complete these two functions in this part.

You may need to implement some helper functions in `assembler.h` first:

`RecognizeNumberValue, NumberToAssemble, ConvertBin2Hex`.

Test

Congratulations! Now you should have finished the tiny LC3 assembler!

Make sure you are in the root directory (which contains Makefile), then you can see the helper information of the assembler by typing command:

```
make  # generate executable file
./assembler -h
```

The output should be:

```
This is a simple assembler for LC-3.

Usage
./assembler [OPTION] ... [FILE] ...

Options
-h : print out help information
-f : the path for the input file
-e : print out error information
-o : the path for the output file
-s : hex mode
```

To assemble an `input.asm` file into `output.bin`, type command (still in root directory):

```
./assembler -f input.asm -o output.bin
```

We have provide you with the testcases in the `test` subdirectory. Your `assembler` must be able to take in `asm` file in `testcases` folder, and output corresponding bin file in `expected` folder.

For example, you can do:

```
mkdir /test/actual
./assembler -f ./test/testcases/testcase1.asm -o ./test/actual/testcase1.bin
diff ./test/actual/testcase1.bin ./test/expected/testcase1.bin
```

If the two files are identical, then diff will output nothing, which means your assembler works.