ICS LAB S

Introduction

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

If you choose to use the framework we provide:

- Your task is to replace all `TO BE DONE` with the correct code.(`TODO` **are not required!**)
- You'll need to learn how to use `Cmake`.
- We recommend you to finish this lab in Linux(vlab is a good option)

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

You may refer to Appendix A in the textbook for more details.

Overview

```
.
├── CMakeLists.txt      # CMakeLists to be used for this lab
├── include
│   ├── common.h
│   ├── memory.h
│   ├── register.h
│   └── simulator.h
└── src
    ├── main.cpp
    ├── memory.cpp
    ├── register.cpp
    └── simulator.cpp
```

You'll need to complete all `TO BE DONE` in `main.cpp`、`memory.cpp` and `simulator.cpp`.(`TODO` **are not required!**)

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_Name_labS.pdf`

CMake

In this lab, we compile our code with the help of CMake.

type command

```
mkdir build
cd build
cmake ..
make
```

will do:

- create a new directory called `build`
- navigate into that directory
- run `cmake` on the parent directory to generate the build instructions
- use the `make` command to build the project

`lc3simulator` is the executable file of our simulator.

Hint

memory.cpp

There are three functions you'll need to complete in `memory.cpp`

- `ReadMemoryFromFile` will read from the file(*.bin) and store the content in `memory`.
- `GetContent` and `[]` operator overloading functions will return the content of the corresponding `address` in memory

simulator.cpp

In `simulator.cpp`, key functions you need to complete are listed here:

- `SignExtend`: Extend the number to corresponding length(`B`). (You may need to learn a little about c++ templates.)
- `UpdateCondRegister`: Update the condition register according to the content of the given register.
- `VM_XX`: The execution of specific instructions. If you get puzzled, you can refer to `VM_ADD`、`VM_BR` and `VM_LD`.Note that `VM_RTI` and `VM_TRAP` are not required. But if you have more interest, You may finish `TODO` in `VM_TRAP`.
- `NextStep`: Execution of single instruction. Call the corresponding function according to the opcode.

main.cpp

In `main.cpp`, you only need to finish the code in while loop.(Actually, maybe only one line) But we still recommend you to read the whole program to understand the running process of the simulator.

Test

We assume you are in the `build` subdirectory (of root directory) and you have built the project after using `cmake ..` and `make`.

You can see the helper information of the simulator by typing command

```
./lc3simulator -h
```

The output should be:

```
LC3 SIMULATOR

Options:
  -h [ --help ]                         Help screen
  -f [ --file ] arg (=input.txt)        Input file
  -r [ --register ] arg (=register.txt) Register Status
  -s [ --single ]                       Single Step Mode
  -b [ --begin ] arg (=12288)           Begin address (0x3000)
  -o [ --output ] arg                   Output file
  -d [ --detail ]                       Detailed Mode
```

To simulate an `input.bin` file and initialize all registers with `register.txt`, type command:

```
./lc3simulator -f input.bin -r register.txt
```

We have provided you with testcases in the `test` subdirectory(of root directory). The input bin files are in `testcases` folder, the register initialization files are in `register` folder and the corresponding output with debug information of every step is in `expected` folder. Besides, the original `asm` files are in `asm` folder for your convenience.

For example, you can do:

```
./lc3simulator -f ../test/testcases/test1.bin -r ../test/register/register1.txt
-d
```

and then compare your output to the expected output.