LabS: Simulator

> In this lab, we will implement a simple LC-3 simulator. The objectives of this experiment are as follows:
>
> - Learn how to install additional dependencies in the code
> - Learn how to use CMake for program compilation
> - Fully understand the LC-3 ISA
>
> > This lab allows the use of frameworks other than the provided one, and there is no restriction on the programming language used. Achieving the same functionality will be considered as completing the lab.

1. Installing the Dependency Library

Type the following commands in your terminal.(`Linux OS`)

```
sudo apt-get update
sudo apt-get install libboost-all-dev -y
```

2. Compilation Process

- The file structure should be as follows.

```
[16:56:17][voyage]~/project/labS$ tree . -f
.
├── ./CMakeLists.txt
├── ./include
│   ├── ./include/common.h
│   ├── ./include/memory.h
│   ├── ./include/register.h
│   └── ./include/simulator.h
└── ./src
    ├── ./src/main.cpp
    ├── ./src/memory.cpp
    ├── ./src/register.cpp
    └── ./src/simulator.cpp
```

- Make sure you are in the root directory of the project and type the following commands in your terminal.

```
mkdir build && cd build
cmake ..
```

If everything goes correct, compilation files will be generated in the current directory, and you will see output similar to the following.

```
[17:00:16][voyage]~/project/labS/build$ cmake ..
-- The C compiler identification is GNU 11.4.0
-- The CXX compiler identification is GNU 11.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found Boost: /home/voyage/anaconda3/lib/cmake/Boost-1.82.0/BoostCo
-- Configuring done
-- Generating done
-- Build files have been written to: /home/voyage/project/labS/build
```

- Type the following command in your terminal.

```
make
```

This command will generate an executable file named `lc3simulator` in the current directory.

- Explanation of the command-line arguments for the executable file.

```
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
```

- A possible example of a run command.

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

3. Introduction to the Lab Framework

The lab framework is mainly composed of four files: `main.cpp`, `memory.cpp`, `register.cpp`, `simulator.cpp`.

- `main.cpp`: The main function file, which includes the definition of command-line arguments, the initialization of the simulator, and the iteration process.
- `memory.cpp`: Memory definition file, with defines the initialization and read/write operations of memory.
- `register.cpp`: Register definition file, wih overloads the register output operations (the initialization process of the registers is implemented in `simulator.cpp`).
- `simulator.cpp`: Simulator file, wich completes the initialization of the simulator, defines the step-by-step simulation function, and implements the LC-3 ISA.

4. Lab Tasks

- Thoroughly read and understand the code framework, complete the **`TO BE DONE`** sections, or independently implement the LC-3 simulator to achieve simulation functionality.
  - One or more programs may be run during the inspection.
  - TA may ask questions about certain parts of the code framework.
- A lab report that reflects your understanding of the lab.
  - There are no fixed format requirements.
  - It should reflect your thoughts or suggestions about the lab.

> Correctness for 50% and the report for other 50%.

5. Submission

- Your submission should be structured as shown below:

```
PB********_Name_labs.zip
 ├─── ./CMakeLists.txt
 ├─── ./PB********_Name_report.pdf
 ├─── ./include
 |    ├─── ./include/common.h
 |    ├─── ./include/memory.h
 |    ├─── ./include/register.h
 |    └─── ./include/simulator.h
 └─── ./src
      ├─── ./src/main.cpp
      ├─── ./src/memory.cpp
      ├─── ./src/register.cpp
      └─── ./src/simulator.cpp
```

6. FAQ

> We encourage asking questions to the TAs, but make sure you have trid it yourself first before asking.

(1) How to have your own Linux operating system

- Install a Linux physical operating system
- Use the school's **vlab service**
- Use **WSL (Windows Subsystem for Linux)**
- Use **Hyper-V**

(2) How to resolve network issues during downloading

- Change the software repository for your distribution： **mirros.ustc.edu.cn**
- ~~*Use magic*~~

(3) How to start translating the instructions

- You can refer to the already provided `ADD` / `BR` / `LD` instruction, it may be help you

.....

7. Postscript

- Thanks to the 2021 course TA **Chivier Humber** for his contributions to the lab framework.
- Rambling notes on this lab：
  - Personally, I feel that the difficulty of this lab is less than labA.
  - The process of installing the Linux operating system is also a learning experience. It will be a part of your university life for at least the next three years, so do not avoid it.