ICS-H Lab-S Report

PB22020514 郭东昊 2024 年 1 月 30 日

Purpose

The purpose of the experiment is to implement a LC-3 simulator and test it with the output of labA. The simulator should support SingleStep mode and have easy-to-check values for registers and memory. The program should handle illegal opcodes, access control violations, privilege mode violations, and other issues.

Principle

It uses the Boost library to parse command-line options. The simulator can be controlled by various command-line arguments.The boost::program_options library is used to define and parse these options.

The simulator then initializes a virtual machine with the specified starting address, input file, and register status file.If the setm option is provided, the simulator parses the value and sets the specified memory address to the given value.

The simulator then enters a loop, executing one step at a time until a halt instruction is encountered. If single-step mode is enabled, it waits for user input before each step. If detailed mode is enabled, it prints the register status after each step.

Finally, the simulator outputs the memory status to a file and prints the final register status and cycle count.

Procedures

There are several problems I met when filling the blanks of this program.

The C++ Language

Before doing this experiment and Lab A, I have never had a try on C++. The concept of class, stream, etc. are so unfamiliar to me.

The C++ STL

As I began to do my work from the **assembler.h,** I found myself confused by the huge amount of “<>”s “#include”ed at the head of this **head** file. Luckily, although I had to start from knowing nothing, I wasn’t required to start from scratch. With the

help doc written by our lovely TA, I read the code given, and got it through. By searching on CSDN, I learned how to use the functions I needed from the STL to complete my mission. After completing the program I got quite a sense of achievement, thanking for all our TAs had done for us students.

The Boost

I was so upset about my time wasted on building a C++ environment with the famous library Boost using CMake. Although I had TRIED a thousand times using thousands of billions methods to include the Boost library I installed, which cost me nearly one half month, I didn't get it through. Finally, I turned to use the VS instead of the CMake to run my program.

Result

Normal execution of the Assembler

For the source file "**Lab4.bin**",Input the command:

-f "E:/LC-3/Lab4.bin" -d -o "E:/LC-3/Lab4_out.txt" -m 0x3100=0x0004

The output of the program reports a success:

```
R4 = 0, R5 = 310a, R6 = 4000, R7 = 3005
COND[NZP] = 001
PC = 3005

TRAP
HALT
R0 = 4, R1 = f, R2 = 1, R3 = 1
R4 = 0, R5 = 310a, R6 = 4000, R7 = 3005
COND[NZP] = 001
PC = 0

----------------------HALT----------------------
Register final status:
R0 = 4, R1 = f, R2 = 1, R3 = 1
R4 = 0, R5 = 310a, R6 = 4000, R7 = 3005
COND[NZP] = 001
PC = 0

cycle = 1de

E:\vscode\LAB_S_Attachment\src\x64\Debug\lc3simulator.exe (进程 6440)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

And the outputfile "**Lab4_out.txt**" shows the memory after the program executed:

```
256   M[30ff]:    0000
257   M[3100]:    0004
258   M[3101]:    0002
259   M[3102]:    0003
260   M[3103]:    000b
261   M[3104]:    000a
262   M[3105]:    0008
263   M[3106]:    0009
264   M[3107]:    000d
265   M[3108]:    000c
266   M[3109]:    000e
267   M[310a]:    000f
268   M[310b]:    0000
```

It is the correct output of Lab4: Baguenaudier

Single Step Mode

In the main loop of “while”, add a getchar() to implement the Single Step Mode.

```
// Single step
if (gIsSingleStepMode) {
    std::cout << "Press any key to continue..." << std::endl;
    getchar();
}
```

ACV violation

In class virtual_machine_tp, I set a bool variable **IsUserMode** to indicate if the virtual machine is in UserMode. (Actually at first I set a PSR, but that was canceled by me because it was not necessary as I am using C++, a high level language, to implement the little computer built on electronics.)

```
class virtual_machine_tp {
    public:
    register_tp reg;
    memory_tp mem;
    bool IsUserMode = true;
```

Then in the implementation of each instruction, I check the **IsUserMode** and the address. If not qualified, the simulator throws the “**ACV violation**” Error message.

```
if(IsUserMode && (reg[R_PC] + pc_offset) > 0xFDFF || (reg[R_PC] + pc_offset) < 0x3000){
    throw MyError("ACV violation");
}
mem[reg[R_PC] + pc_offset] = reg[sr];
```

TRAP routine

With C functions such like getchar() and putchar(), the implementation of TRAP routine is quite easy with a few “if”s.

```
    // PUTS
    int16_t* c = &mem[reg[R_R0]];
    while (*c)
    {
        putchar(*c);
        ++c;
    }
    std::cout << "This is the output of <TRAP x22>, press any key to continue......" << std::endl;
    getchar();
}
else if (trapnum == 0x23)
{
    // IN
    printf("<TRAP x23>: ");
    reg[R_R0] = getchar();
}
else if (trapnum == 0x25)
{
    // HALT
    printf("HALT\n");
```

Summary

In this experiment, I was required to implement a LC-3 simulator and test it using the output of labA. The simulator needed to support SingleStep mode and provide easy-to-check values for registers and memory.Additionally, if the setm option was provided, the simulator would parse the value and set the specified memory address accordingly.

Throughout the experiment, several challenges were encountered. As for the results, the normal execution of the Assembler for the provided source file "Lab4.bin" was successful. The program generated the expected output and the memory after execution matched the desired result. ACV (Access Control Violation) was also implemented, where the simulator checked the IsUserMode variable and address validity to throw an "ACV violation" error message if necessary.

In conclusion,during this Lab, challenges were overcome through research and assistance, and the experiment was successfully completed. Gratitude is extended to all the TAs for their valuable support throughout the process.