LabS 模拟器

实验内容

- 使用C++实现LC3模拟器，输入二进制机器码，输出这一串二进制机器码执行结果

代码架构

- 算法实现的基本思想就是逐行进行扫描，并把一行16个二进制位按照不同的定义进行译码，并对相应的寄存器作相应的运算
- 在二进制机器码执行过程（即Nextstep函数）当中，会出现符号位扩展以及更新寄存器的状态的操作，这些操作也需要进行相应描述

核心代码

SignExtend函数

```
inline T SignExtend(const T x) {
    // Extend the number
    if(x & (1 << (B - 1))) return  x | (-1 << B);
    else return x;
}
```

UpdateCondRegister函数

```
void virtual_machine_tp::UpdateCondRegister(int regname) {
    // Update the condition register
    if(reg[regname] < 0) reg[R_COND] = 0b100;
    else if(reg[regname] == 0) reg[R_COND] = 0b010;
    else reg[R_COND] = 0b001;
}
```

执行操作函数（这里以BR为例）

```
void virtual_machine_tp::VM_BR(int16_t inst) {
    int16_t pc_offset = SignExtend<int16_t, 9>(inst & 0x1FF);
    int16_t cond_flag = (inst >> 9) & 0x7;
    if (gIsDetailedMode) {
        std::cout << reg[R_PC] << std::endl;
        std::cout << pc_offset << std::endl;
    }
    if (cond_flag & reg[R_COND]) {
        reg[R_PC] += pc_offset;
    }
}
```

主函数的关键部分(循环终止)

```
int halt_flag = true;
    int time_flag = 0;
    while(halt_flag) {
        // Single step
        halt_flag = virtual_machine.NextStep();
        if (gIsDetailedMode)
            std::cout << virtual_machine.reg << std::endl;
        ++time_flag;
    }
    std::cout << virtual_machine.reg << std::endl;
    std::cout << "cycle = " << time_flag << std::endl;
    std::ofstream result;
    result.open(gOutputFileName);
    if (result.is_open()) {
        result << virtual_machine.reg << std::endl;
        result << "cycle = " << time_flag << std::endl;
    }
```

读入数据函数

```
inline int16_t TranslateInstruction(std::string &line) {
    // TODO: translate hex mode
    int16_t result = 0;
    if (line.size() == kInstructionLength) {
        for (int index = 0; index < kInstructionLength; ++index) {
            result = (result << 1) | (line[index] & 1);
        }
    }
    return result;
}
```

模拟器执行函数

```
int16_t virtual_machine_tp::NextStep() {
    int16_t current_pc = reg[R_PC];
    reg[R_PC]++;
    int16_t current_instruct = mem[current_pc];
    int opcode = (current_instruct >> 12) & 15;

    switch (opcode) {
        case O_ADD:
        if (gIsDetailedMode) {
            std::cout << "ADD" << std::endl;
        }
        VM_ADD(current_instruct);
        break;
        case O_AND:
            if (gIsDetailedMode) {
            std::cout << "AND" << std::endl;
        }
        VM_AND(current_instruct);
        break;
        case O_BR:
```

```
            if (gIsDetailedMode) {
            std::cout << "BR" << std::endl;
        }
        VM_BR(current_instruct);
        break;
        case O_JMP:
            if (gIsDetailedMode) {
            std::cout << "JMP" << std::endl;
        }
        VM_JMP(current_instruct);
        break;
        case O_JSR:
            if (gIsDetailedMode) {
            std::cout << "JSR" << std::endl;
        }
        VM_JSR(current_instruct);
        break;
        case O_LD:
            if (gIsDetailedMode) {
            std::cout << "LD" << std::endl;
        }
        VM_LD(current_instruct);
        break;
        case O_LDI:
            if (gIsDetailedMode) {
            std::cout << "LDI" << std::endl;
        }
        VM_LDI(current_instruct);
        break;
        case O_LDR:
            if (gIsDetailedMode) {
            std::cout << "LDR" << std::endl;
        }
        VM_LDR(current_instruct);
        break;
        case O_LEA:
            if (gIsDetailedMode) {
            std::cout << "LEA" << std::endl;
        }
        VM_LEA(current_instruct);
        break;
        case O_NOT:
            if (gIsDetailedMode) {
            std::cout << "NOT" << std::endl;
        }
        VM_NOT(current_instruct);
        break;
        case O_RTI:
            if (gIsDetailedMode) {
            std::cout << "RTI" << std::endl;
        }
        VM_RTI(current_instruct);
        break;
        case O_ST:
            if (gIsDetailedMode) {
            std::cout << "ST" << std::endl;
        }
        VM_ST(current_instruct);
```

```
            break;
            case O_STI:
                if (gIsDetailedMode) {
                std::cout << "STI" << std::endl;
            }
            VM_STI(current_instruct);
            break;
            case O_STR:
                if (gIsDetailedMode) {
                std::cout << "STR" << std::endl;
            }
            VM_STR(current_instruct);
            break;
            case O_TRAP:
            if (gIsDetailedMode) {
                std::cout << "TRAP" << std::endl;
            }
            if ((current_instruct & 0xFF) == 0x25) {
                reg[R_PC] = 0;
            }
            VM_TRAP(current_instruct);
            break;
            default:
            VM_RTI(current_instruct);
            break;
        }

        if (current_instruct == 0) {
            // END
            // TODO: add more detailed judge information
            return 0;
        }
        return reg[R_PC];
    }
```

实验总结

通过本次实验，我了解了LC3模拟器的基本执行操作，同时也对计算机内部指令集有了更深刻的认识，加深了对计算机指令执行流程的理解。