

实验报告——labs simulator

PB20151793 宋玮

本实验是编写一个模拟器，实现机器码到具体操作的实现。

总共分为 4 个部分，simulator，memory，register 和 main.

其中 main.cpp 中总共列举了 7 中不同运行功能，包括 help，file，register，single，begin，output 和 detail。

(1) 首先看 memory 部分，我们需要把文件所给的机器码准换成实际信息存入 memory 中。如下所示：

```
11 namespace virtual_machine_nsp {
12 void memory_tp::ReadMemoryFromFile(std::string filename, int beginning_address) {
13     // Read from the file
14     // TO BE DONE
15     std::ifstream memoryfile; // 读入memoryfile
16     memoryfile.open(filename); // 打开名字对应文件
17     int ptr = beginning_address; // 起始地址
18     int line_count = std::count(std::istreambuf_iterator<char>(memoryfile), std::istreambuf_iterator<char>(), '\n'); // 查找
19     memoryfile.close(); // 结束该操作
20     memoryfile.open(filename);
21     char instru[17]; // 16位指令
22     for (int i = 0; i < line_count; i++) {
23         memoryfile >> instru;
24         memory[ptr] = 0;
25         for (int j = 0; instru[j]; j++) {
26             memory[ptr] += (instru[j] - 48) * pow(2, 16 - j - 1);
27         }
28         ptr++;
29     }
30     memoryfile.close();
31 }
```

其次要定义对 memory 进行存取操作的函数。如下：

```
33 int16_t memory_tp::GetContent(int address) const {
34     // get the content
35     // TO BE DONE
36     return memory[address];
37 }
38
39 int16_t& memory_tp::operator[](int address) {
40     // get the content
41     // TO BE DONE
42     return memory[address];
43 }
44 }; // virtual machine namespace
```

至此，memory 部分完成。

(2) simulator 部分。

首先是位扩展函数，分为 x 为正数和 x 为负数两种情况。如下：

```
10 namespace virtual_machine_nsp {
11 template <typename T, unsigned B>
12 inline T SignExtend(const T x) { // 对x进行扩展
13     // Extend the number
14     // TO BE DONE
15     int16_t cnt = B;
16     int16_t m = 1;
17     T high; // high 最高位
18     int16_t y;
19     for (int i = 1; i < cnt; i++) {
20         m = m << 1;
21     }
22     high = x & m;
23     if (high == 0) { // 正数直接
24         y = x;
25         return y;
26     }
27     else {
28         T temp = ~(~(T)pow(2, B) - 1); // 负数前补1
29         y = x | temp;
30         return y;
31     }
32 }
33 }
```

其次是状态寄存器的变化，即 nzp
引起变化的则是目的寄存器里写入的值的正负。

如下：

```
35 void virtual_machine_tp::UpdateCondRegister(int regname) {
36     // Update the condition register
37     // TO BE DONE
38     if (reg[regname] > 0) reg[R_COND] = 0b001;
39     else if (reg[regname] == 0) reg[R_COND] = 0b010; //nzp
40     else reg[R_COND]=0b100;
41 }
```

然后是对 ADD，AND，NOT 等各项指令执行过程的补充，下面只列举 LDR 操作：

```
147 void virtual_machine_tp::VM_LDR(int16_t inst) {
148     // TO BE DONE
149     int dr = (inst >> 9) & 0x7;
150     int baser = (inst >> 6) & 0x7;
151     int16_t offset6 = SignExtend<int16_t, 6>(inst & 0b111111);
152     reg[dr] = mem[reg[baser] + offset6];
153     UpdateCondRegister(dr);
154 }
155 }
```

其余操作过程类似。

最后是虚拟机执行状态的变换：

```
259 int16_t virtual_machine_tp::NextStep() {
260     int16_t current_pc = reg[R_PC];
261     reg[R_PC]++;
262     int16_t current_instruct = mem[current_pc];
263     int opcode = (current_instruct >> 12) & 15;
264
265     switch (opcode) {
266     case 0_ADD:
267         if (gIsDetailedMode) {
268             std::cout << "ADD" << std::endl;
269         }
270         VM_ADD(current_instruct);
271         break;
272     case 0_AND:
273         // TO BE DONE
274         if (gIsDetailedMode) {
275             std::cout << "AND" << std::endl;
276         }
277         VM_AND(current_instruct);
278         break;
279     case 0_BR:
280         // TO BE DONE
281         if (gIsDetailedMode) {
282             std::cout << "BR" << std::endl;
283         }
284         VM_BR(current_instruct);
285         break;
```

switch 通过对指令 opcode 的选择，来完成不同指令和状态转换，图中只截取了 ADD,AND,BR。其余类似。

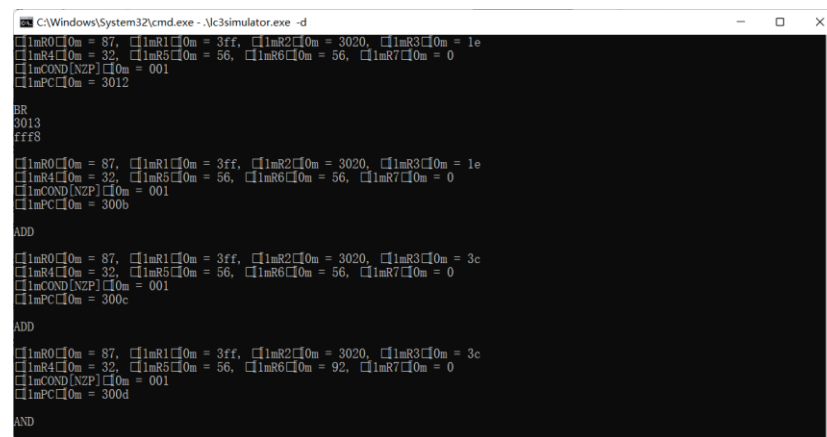
(3) main 部分。

main.cpp 中总共列举了 7 中不同运行功能，包括 help，file，register，single，begin，output 和 detail。

假设你要以每一次回车输出每一次执行的 detail 信息。可以在 halt-flag 部分执行这样的操作：

```
62 while(halt_flag) {
63     // Single step
64     // TO BE DONE
65     halt_flag = (virtual_machine.NextStep())&&((c=getchar())=='\n');
66     if (gIsDetailedMode)
67         std::cout << virtual_machine.reg << std::endl;
68     ++time_flag;
69 }
```

效果图:



其余功能只需稍加修改。

(4) register 部分比较简单，主要是编码了寄存器和状态，并且打印输出。

```
11 namespace virtual_machine_nsp {
12     std::ostream& operator<<(std::ostream& os, const register_tp& reg) { //重载, 类似cout
13         os << "\e[1mR0\e[0m = " << std::hex << reg[R_R0] << ", ";
14         os << "\e[1mR1\e[0m = " << std::hex << reg[R_R1] << ", ";
15         os << "\e[1mR2\e[0m = " << std::hex << reg[R_R2] << ", ";
16         os << "\e[1mR3\e[0m = " << std::hex << reg[R_R3] << std::endl;
17         os << "\e[1mR4\e[0m = " << std::hex << reg[R_R4] << ", ";
18         os << "\e[1mR5\e[0m = " << std::hex << reg[R_R5] << ", ";
19         os << "\e[1mR6\e[0m = " << std::hex << reg[R_R6] << ", ";
20         os << "\e[1mR7\e[0m = " << std::hex << reg[R_R7] << std::endl;
21         os << "\e[1mCOND[NZP]\e[0m = " << std::bitset<3>(reg[R_COND]) << std::endl;
22         os << "\e[1mPC\e[0m = " << std::hex << reg[R_PC] << std::endl;
23         return os;
24     }
25 } // virtual machine namespace
```

总结:在终端输入 `cmake.exe -G"MinGW Makefiles" .\CMakeLists.txt` 和 `mingw32-make.exe`, 若没有错误, 即可编译成功。测试机器码放在同目录 `input.txt` 中。之后可生成 `lc3simulator.exe` 可执行文件。