ICS LabS 实验报告

张艺耀 PB2011630

最终文件目录如下

深度为2的树型目录 `tree -L 2`

```
.
├── CMakeCache.txt
├── CMakeFiles
|   ├── 3.22.1
|   ├── CMakeDirectoryInformation.cmake
|   ├── CMakeError.log
|   ├── CMakeOutput.log
|   ├── CMakeTmp
|   ├── Makefile.cmake
|   ├── Makefile2
|   ├── TargetDirectories.txt
|   ├── cmake.check_cache
|   ├── lc3simulator.dir
|   └── progress.marks
├── CMakeLists.txt
├── Makefile
├── cmake_install.cmake
├── include
|   ├── _memory.h
|   ├── common.h
|   ├── register.h
|   └── simulator.h
├── input.txt
```

```
├── inputs
|   └── input.txt
├── lc3simulator
├── output.txt
└── src
    ├── main.cpp
    ├── memory.cpp
    ├── register.cpp
    └── simulator.cpp
```

前置条件

- 安装Cmake
- 安装boost

`port install boost`

boost相关头文件默认存在于`/opt/local/include/`目录下

故需要在c_cpp_properties.json中includepath：

```
//c_cpp_properities.json
{
    "configurations": [
        {
            "name": "Mac",
            "includePath": [
                "${workspaceFolder}/**",
                "/opt/local/include/"
            ],
            "defines": [],
            "macFrameworkPath": [
```

```
12
    "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk/System/Library/Frameworks"
13              ],
14              "compilerPath": "/usr/bin/clang",
15              "cStandard": "c17",
16              "cppStandard": "c++11",
17              "intelliSenseMode": "macos-clang-arm64"
18          }
19      ],
20      "version": 4
21  }
```

代码分析与阅读

common.h

- 头文件的包含，全局变量的声明。

```
1  //common.h
2  #pragma once
3
4  #include <iostream>
5  #include <fstream>
6  #include <cstdio>
7  #include <array>
8  #include <vector>
9  #include <cmath>
10 #include <climits>
11 #include <cstdlib>
12 #include <string>
13 #include <cstring>
14 #include <algorithm>
15
16 // Boost library
17 #include <boost/program_options.hpp>
```

```
18
19 // Application global variables
20 extern bool gIsSingleStepMode;  //  单步模式
21 extern bool gIsDetailedMode;  //  细节模式
22 extern std::string gInputFileName;
23 extern std::string gRegisterStatusFileName;
24 extern std::string gOutputFileName;
25 extern int gBeginningAddress;
```

_memory.h

类memory_tp的对象为长为65535的int16_t类型数组，其构造函数memory_tp将数组清零。

该类的方法为从文件中读取内存和得到某地址的内存。

由于助教给的原文件名为memory.h 与某库文件冲突，故改名。

```
1  //_memory.h
2  #include "common.h"
3
4  namespace virtual_machine_nsp {
5  const int kInstructionLength = 16;
6
7  inline int16_t TranslateInstruction(std::string &line) {
8      // TODO: translate hex mode
9      short num = 0; if(line.size() != 16) return 0;
10     for (int i = 0; i < kInstructionLength; i++)
11         num = (num << 1) | (line[i] & 1);
12     return num;
13 }
14
15 const int kVirtualMachineMemorySize = 0xFFFF;
16
17 class memory_tp {
18     private:
```

```
    int16_t memory[kVirtualMachineMemorySize];

    public:
    memory_tp() {
        memset(memory, 0, sizeof(int16_t) * kVirtualMachineMemorySize);
    }
    // Managements
    void ReadMemoryFromFile(std::string filename, int
beginning_address=0x3000);
    int16_t GetContent(int address) const;
    int16_t& operator[](int address);
};

}; // virtual machine nsp
```

register.h

- 声明virtual_machine_nsp名字空间下的寄存器数和寄存器枚举类型，数出运算符重载。

```
//register.h
#include "common.h"
namespace virtual_machine_nsp {
    const int kRegisterNumber = 10;
    enum RegisterName {
        R_R0 = 0,
        R_R1,
        R_R2,
        R_R3,
        R_R4,
        R_R5,
        R_R6,
        R_R7,
        R_PC, // 8
        R_COND // 9
    };
```

```
17
18      typedef std::array<int16_t, kRegisterNumber> register_tp;
19      std::ostream& operator<<(std::ostream& os, const register_tp& reg);
20  } // virtual machine namespace
```

simulator.h

- 声明virtual_machine_nsp名字空间下的操作数枚举类型。声明virtual_machine_tp类，其成员为寄存器Array类型数组和内存（memory）。方法为各个操作数对应的操作和构造函数，执行函数等等。

```
1   //simulator.h
2   #pragma once
3
4   #include "common.h"
5   #include "register.h"
6   #include "_memory.h"
7
8   namespace virtual_machine_nsp {
9
10  enum kOpcodeList {
11      O_ADD = 0b0001,
12      O_AND = 0b0101,
13      O_BR  = 0b0000,
14      O_JMP = 0b1100,
15      O_JSR = 0b0100,
16      O_LD  = 0b0010,
17      O_LDI = 0b1010,
18      O_LDR = 0b0110,
19      O_LEA = 0b1110,
20      O_NOT = 0b1001,
21      O_RTI = 0b1000,
22      O_ST  = 0b0011,
23      O_STI = 0b1011,
24      O_STR = 0b0111,
25      O_TRAP = 0b1111
```

```
26 };
27
28 enum kTrapRoutineList {
29 };
30
31 class virtual_machine_tp {
32     public:
33     register_tp reg;
34     memory_tp mem;
35
36     // Instructions
37     void VM_ADD(int16_t inst);
38     void VM_AND(int16_t inst);
39     void VM_BR(int16_t inst);
40     void VM_JMP(int16_t inst);
41     void VM_JSR(int16_t inst);
42     void VM_LD(int16_t inst);
43     void VM_LDI(int16_t inst);
44     void VM_LDR(int16_t inst);
45     void VM_LEA(int16_t inst);
46     void VM_NOT(int16_t inst);
47     void VM_RTI(int16_t inst);
48     void VM_ST(int16_t inst);
49     void VM_STI(int16_t inst);
50     void VM_STR(int16_t inst);
51     void VM_TRAP(int16_t inst);
52
53     // Managements
54     virtual_machine_tp() {} //无参数构造函数
55     virtual_machine_tp(const int16_t address, const std::string
   &memfile, const std::string &regfile);
56     void UpdateCondRegister(int reg);
57     void SetReg(const register_tp &new_reg);
58     int16_t NextStep();
59 };
60
```

```
61 }; // virtual machine namespace
```

main.cpp

- 头文件的包含，全局变量的定义，默认初始地址的定义。

```
1  #include "simulator.h"
2
3  using namespace virtual_machine_nsp;
4  namespace po = boost::program_options;
5
6  bool gIsSingleStepMode = false;
7  bool gIsDetailedMode = false;
8  std::string gInputFileName = "input.txt";
9  std::string gRegisterStatusFileName = "register.txt";
10 std::string gOutputFileName = "";
11 int gBeginningAddress = 0x3000;
```

- Linux Bash GNU下的转义字符\e

格式控制：`\e[<格式代码>m`

代码十进制	作用
0	清除所有格式（常用在格式控制末尾，以免对后序字符串造成影响）
1	加粗 与格式 2 冲突
2	字体变暗 与格式 1 冲突
3	斜体
4	下划线
5	呼吸闪烁（但有的机器上没效果）
6	同上
7	反显（背景色当前景色，前景色当背景色）
8	隐形（字符仍然存在，可以选中，只是看不到）
9	删除线

EX:

```c
#include<stdio.h>
int main()
{
        int i;
        printf("输出格式设置：0-9\n");
        for(i=0;i<=9;++i){
                printf("%d: \e[%dmHello,World!\e[0m\n",i,i);
        }
        return 0;
}
```

颜色控制：`\e[<颜色代码>m` *对于同一颜色，背景色代码=前景色代码 + 10*

前景色代码	颜色	背景色代码
30	黑色	40
31	红色	41
32	绿色	42
33	黄色	43
34	蓝色	44
35	紫色	45
36	青色	46
37	白色 *亮灰色*	47

EX:

```
#include<stdio.h>
int main()
{
    int i;
    printf("前景色设置：30-37\n");
    for(i=30;i<=37;++i){
        printf("%d: \e[%dmHello World!\e[0m\n",i,i);  //\e[0m不可或缺
    }
    printf("背景色设置：40-47\n");
    for(i=40;i<=47;++i){
        printf("%d: \e[%dmHello World!\e[0m\n",i,i);
    }
    return 0;
}
```

- boost::program_options

  program options是一系列 `pair<name,value>` 组成的选项列表,它允许程序通过命令行或配置文件来读取这些参数选项。

```
int main(int argc, char **argv) {
    po::options_description desc{"\e[1mLC3 SIMULATOR\e[0m\n\n\e[1mOptions\e[0m"};   //  选项描述器
    desc.add_options()      //为选项描述器增加选项 参数为key，value类型，该选项的描述
        ("help,h", "Help screen")
                            //
        ("file,f", po::value<std::string>()->default_value("input.txt"), "Input file")              //
        ("register,r", po::value<std::string>()->default_value("register.txt"), "Register Status") //
        ("single,s", "Single Step Mode")
                            //
        ("begin,b", po::value<int>()->default_value(0x3000), "Begin address (0x3000)")
        ("output,o", po::value<std::string>()->default_value(""), "Output file")
        ("detail,d", "Detailed Mode");

    po::variables_map vm;   //  选项存储器
    store(parse_command_line(argc, argv, desc), vm);
 //parse_command_line()对输入的选项做解析 store()将解析后的结果存入选项存储器
    notify(vm); //  更新外部变量

    if (vm.count("help")) {
        //options_description对象支持流输出，  会自动打印所有的选项信息
        std::cout << desc << std::endl;
        return 0;
    }
    if (vm.count("file")) {
        //variables_map(选项存储器)是std::map的派生类,可以像关联容器一样使用,
        //通过operator[]来取出其中的元素.但其内部的元素类型value_type是boost::any,
        //用来存储不确定类型的参数值,必须通过模板成员函数as<type>()做类型转换后,
        //才能获取其具体值.
        gInputFileName = vm["file"].as<std::string>();
```

```
    }
    if (vm.count("register")) {
        gRegisterStatusFileName = vm["register"].as<std::string>();
    }
    if (vm.count("single")) {
        gIsSingleStepMode = true;
    }
    if (vm.count("begin")) {
        gBeginningAddress = vm["begin"].as<int>();
    }
    if (vm.count("output")) {
        gOutputFileName = vm["output"].as<std::string>();
    }
    if (vm.count("detail")) {
        gIsDetailedMode = true;
    }
  ...
}
```

- 执行simulator部分：

```
  virtual_machine_tp virtual_machine(gBeginningAddress, gInputFileName, gRegisterStatusFileName);
    int halt_flag = true;
    int time_flag = 0;
    while(halt_flag) {
        // Single step
        // TO BE DONE
        if (virtual_machine.NextStep() == 0)
            halt_flag = 0;
        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;
    return 0;
```

memory.cpp

- 对memory.h中函数的定义

```
//memory.cpp
#include "common.h"
#include "_memory.h"
namespace virtual_machine_nsp {
    void memory_tp::ReadMemoryFromFile(std::string filename, int beginning_address) {
        // Read from the file
        // TO BE DONE
        std::ifstream input_file;
        std::string s;
        int16_t tmp = 0;
        input_file.open(filename);
        int addr = beginning_address;
        while(getline(input_file, s)){
            for(int i = 0; i < 16; i++)
                tmp += (s[i] - '0') << (15 - i);
            memory[addr] = tmp;
            tmp = 0;
            addr++;
        }
        input_file.close();
    }

    int16_t memory_tp::GetContent(int address) const {
        // get the content
        // TO BE DONE
        return memory[address];
```

```
    }

    int16_t& memory_tp::operator[](int address) {
        // get the content
        // TO BE DONE
        return memory[address];
    }
}; // virtual machine namespace
```

register.cpp

- 打印寄存器值

```
//register.cpp
#include "register.h"

namespace virtual_machine_nsp {
    std::ostream& operator<<(std::ostream& os, const register_tp& reg)
{
        os << "\e[1mR0\e[0m = " << std::hex << reg[R_R0] << ", ";
        os << "\e[1mR1\e[0m = " << std::hex << reg[R_R1] << ", ";
        os << "\e[1mR2\e[0m = " << std::hex << reg[R_R2] << ", ";
        os << "\e[1mR3\e[0m = " << std::hex << reg[R_R3] << std::endl;
        os << "\e[1mR4\e[0m = " << std::hex << reg[R_R4] << ", ";
        os << "\e[1mR5\e[0m = " << std::hex << reg[R_R5] << ", ";
        os << "\e[1mR6\e[0m = " << std::hex << reg[R_R6] << ", ";
        os << "\e[1mR7\e[0m = " << std::hex << reg[R_R7] << std::endl;
        os << "\e[1mCOND[NZP]\e[0m = " << std::bitset<3>(reg[R_COND])
<< std::endl;
        os << "\e[1mPC\e[0m = " << std::hex << reg[R_PC] << std::endl;
        return os;
    }
} // virtual machine namespace
```

simulator.cpp

- 较为关键的一个文件 包含了位扩展函数、条件码更新函数、操作函数、主函数（读取文件，设置初始寄存器和地址值）、设置寄存器函数、执行函数的定义。

```cpp
//simulator.cpp
#include "simulator.h"

namespace virtual_machine_nsp {
template <typename T, unsigned B>
inline T SignExtend(const T x) {
    // Extend the number
    // TO BE DONE
    short i = 1, j = 0xffff;
    if((i << (B - 1)) & x)    //最高位为1
        return x | (j << (B - 1));
    return x;
}

void virtual_machine_tp::UpdateCondRegister(int regname) {
    // Update the condition register
    // TO BE DONE
    if(reg[regname] == 0) reg[R_COND] = 2;
    else if(reg[regname] < 0) reg[R_COND] = 4;
    else reg[R_COND] = 1;
}

void virtual_machine_tp::VM_ADD(int16_t inst) {
    int flag = inst & 0b100000;
    int dr = (inst >> 9) & 0x7; //0b111
    int sr1 = (inst >> 6) & 0x7;    //0b111
    if (flag) { //  立即数模式
        // add inst number
        int16_t imm = SignExtend<int16_t, 5>(inst & 0b11111);
        reg[dr] = reg[sr1] + imm;
    } else {// 寄存器模式
```

```
32          // add register
33          int sr2 = inst & 0x7;
34          reg[dr] = reg[sr1] + reg[sr2];
35      }
36      // Update condition register
37      UpdateCondRegister(dr);
38  }
39
40  void virtual_machine_tp::VM_AND(int16_t inst) {
41      // TO BE DONE
42      int flag = inst & 0b100000;
43      int dr = (inst >> 9) & 0x7; //0b111
44      int sr1 = (inst >> 6) & 0x7;    //0b111
45      if (flag) { //  立即数模式
46          // add inst number
47          int16_t imm = SignExtend<int16_t, 5>(inst & 0b11111);
48          reg[dr] = reg[sr1] & imm;
49      } else {// 寄存器模式
50          // add register
51          int sr2 = inst & 0x7;
52          reg[dr] = reg[sr1] & reg[sr2];
53      }
54      // Update condition register
55      UpdateCondRegister(dr);
56  }
57
58  void virtual_machine_tp::VM_BR(int16_t inst) {
59      int16_t pc_offset = SignExtend<int16_t, 9>(inst & 0x1FF);   //
    inst前9位
60      int16_t cond_flag = (inst >> 9) & 0x7;
61      if (gIsDetailedMode) {
62          std::cout << reg[R_PC] << std::endl;
63          std::cout << pc_offset << std::endl;
64      }
65      if (cond_flag & reg[R_COND]) {
66          reg[R_PC] += pc_offset;
```

```
67      }
68  }
69
70  void virtual_machine_tp::VM_JMP(int16_t inst) {
71      // TO BE DONE
72      int16_t baser = (inst >> 6) & 0x7;
73      reg[R_PC] = baser;
74  }
75
76  void virtual_machine_tp::VM_JSR(int16_t inst) {
77      // TO BE DONE
78      reg[R_R7] = reg[R_PC];
79      int16_t pc_offset = SignExtend<int16_t, 11>(inst & 0x7FF);
80      reg[R_PC] += pc_offset;
81  }
82
83  void virtual_machine_tp::VM_LD(int16_t inst) {
84      int16_t dr = (inst >> 9) & 0x7;
85      int16_t pc_offset = SignExtend<int16_t, 9>(inst & 0x1FF);
86      reg[dr] = mem[reg[R_PC] + pc_offset];
87      UpdateCondRegister(dr);
88  }
89
90  void virtual_machine_tp::VM_LDI(int16_t inst) {
91      // TO BE DONE
92      int16_t dr = (inst >> 9) & 0x7;
93      int16_t pc_offset = SignExtend<int16_t, 9>(inst & 0x1FF);
94      reg[dr] = mem[mem[reg[R_PC] + pc_offset]];
95      UpdateCondRegister(dr);
96  }
97
98  void virtual_machine_tp::VM_LDR(int16_t inst) {
99      // TO BE DONE
100     int16_t dr = (inst >> 9) & 0x7;
101     int16_t baser = (inst >> 6) & 0x7;
102     int16_t offset = SignExtend<int16_t, 6>(inst &0x3F);
```

```
103     reg[dr] = mem[reg[baser] + offset];
104     UpdateCondRegister(dr);
105 }
106 
107 void virtual_machine_tp::VM_LEA(int16_t inst) {
108     // TO BE DONE
109     int16_t dr = (inst >> 9) & 0x7;
110     int16_t pc_offset = SignExtend<int16_t, 9>(inst & 0x1FF);
111     reg[dr] = reg[R_PC] + pc_offset;
112 }
113 
114 void virtual_machine_tp::VM_NOT(int16_t inst) {
115     // TO BE DONE
116     int16_t dr = (inst >> 9) & 0x7;
117     int16_t sr = (inst >> 6) & 0x7;
118     reg[dr] = ~ reg[sr];
119 }
120 
121 void virtual_machine_tp::VM_RTI(int16_t inst) {
122     ; // PASS
123 }
124 
125 void virtual_machine_tp::VM_ST(int16_t inst) {
126     // TO BE DONE
127     int16_t sr = (inst >> 9) & 0x7;
128     int16_t pc_offset = SignExtend<int16_t, 9>(inst & 0x1FF);
129     mem[reg[R_PC] + pc_offset] = reg[sr];
130 }
131 
132 void virtual_machine_tp::VM_STI(int16_t inst) {
133     // TO BE DONE
134     int16_t dr = (inst >> 9) & 0x7;
135     int16_t pc_offset = SignExtend<int16_t, 9>(inst & 0x1FF);
136     mem[mem[reg[R_PC] + pc_offset]] = reg[dr];
137 }
138 
```

```
139 void virtual_machine_tp::VM_STR(int16_t inst) {
140     // TO BE DONE
141     int16_t dr = (inst >> 9) & 0x7;
142     int16_t baser = (inst >> 6) & 0x7;
143     int16_t offset = SignExtend<int16_t, 6>(inst &0x3F);
144     mem[reg[baser] + offset] = reg[dr];
145 }
146 
147 void virtual_machine_tp::VM_TRAP(int16_t inst) {
148     int trapnum = inst & 0xFF;
149     // if (trapnum == 0x25)
150     //     exit(0);
151     // TODO: build trap program
152     if(trapnum == 0x25) exit(0);
153 }
154 
155 virtual_machine_tp::virtual_machine_tp(const int16_t address, const
    std::string &memfile, const std::string &regfile) {
156     // Read memory
157     if (memfile != ""){
158         mem.ReadMemoryFromFile(memfile);
159     }
160 
161     // Read registers
162     std::ifstream input_file;
163     input_file.open(regfile);
164     if (input_file.is_open()) {
165         int line_count = std::count(std::istreambuf_iterator<char>
    (input_file), std::istreambuf_iterator<char>(), '\n');    //返回
    [first, last)范围内等于val的元素数
166         input_file.close();
167         input_file.open(regfile);
168         if (line_count >= 8) {
169             for (int index = R_R0; index <= R_R7; ++index) {
170                 input_file >> reg[index];
171             }
```

```
172         } else {
173             for (int index = R_R0; index <= R_R7; ++index) {
174                 reg[index] = 0;
175             }
176         }
177         input_file.close();
178     } else {
179         for (int index = R_R0; index <= R_R7; ++index) {
180             reg[index] = 0;
181         }
182     }
183 
184     // Set address
185     reg[R_PC] = address;
186     reg[R_COND] = 0;
187 }
188 
189 void virtual_machine_tp::SetReg(const register_tp &new_reg) {
190     reg = new_reg;
191 }
192 
193 int16_t virtual_machine_tp::NextStep() {
194     int16_t current_pc = reg[R_PC];
195     reg[R_PC]++;
196     int16_t current_instruct = mem[current_pc];
197     int opcode = (current_instruct >> 12) & 15; //0b1111
198 
199     switch (opcode) {
200         case O_ADD:
201             if (gIsDetailedMode) {
202                 std::cout << "ADD" << std::endl;
203             }
204             VM_ADD(current_instruct);
205             break;
206         case O_AND:
207         // TO BE DONE
```

```
208             if (gIsDetailedMode) {
209                 std::cout << "AND" << std::endl;
210             }
211             VM_AND(current_instruct);
212             break;
213         case O_BR:
214         // TO BE DONE
215             if (gIsDetailedMode) {
216                 std::cout << "BR" << std::endl;
217             }
218             VM_BR(current_instruct);
219             break;
220         case O_JMP:
221         // TO BE DONE
222             if (gIsDetailedMode) {
223                 std::cout << "JMP" << std::endl;
224             }
225             VM_JMP(current_instruct);
226             break;
227         case O_JSR:
228         // TO BE DONE
229             if (gIsDetailedMode) {
230                 std::cout << "JSR" << std::endl;
231             }
232             VM_JSR(current_instruct);
233             break;
234         case O_LD:
235         // TO BE DONE
236             if (gIsDetailedMode) {
237                 std::cout << "LD" << std::endl;
238             }
239             VM_LD(current_instruct);
240             break;
241         case O_LDI:
242         // TO BE DONE
243             if (gIsDetailedMode) {
```

```
244                 std::cout << "LDI" << std::endl;
245             }
246             VM_LDI(current_instruct);
247             break;
248         case O_LDR:
249         // TO BE DONE
250             if (gIsDetailedMode) {
251                 std::cout << "LDR" << std::endl;
252             }
253             VM_LDR(current_instruct);
254             break;
255         case O_LEA:
256         // TO BE DONE
257             if (gIsDetailedMode) {
258                 std::cout << "LEA" << std::endl;
259             }
260             VM_LEA(current_instruct);
261             break;
262         case O_NOT:
263         // TO BE DONE
264             if (gIsDetailedMode) {
265                 std::cout << "NOT" << std::endl;
266             }
267             VM_NOT(current_instruct);
268             break;
269         case O_RTI:
270         // TO BE DONE
271             if (gIsDetailedMode) {
272                 std::cout << "RTI" << std::endl;
273             }
274             VM_RTI(current_instruct);
275             break;
276         case O_ST:
277         // TO BE DONE
278             if (gIsDetailedMode) {
279                 std::cout << "ST" << std::endl;
```

```
280             }
281             VM_ST(current_instruct);
282             break;
283         case O_STI:
284         // TO BE DONE
285             if (gIsDetailedMode) {
286                 std::cout << "STI" << std::endl;
287             }
288             VM_STI(current_instruct);
289             break;
290         case O_STR:
291         // TO BE DONE
292             if (gIsDetailedMode) {
293                 std::cout << "STR" << std::endl;
294             }
295             VM_STR(current_instruct);
296             break;
297         case O_TRAP:
298         if (gIsDetailedMode) {
299             std::cout << "TRAP" << std::endl;
300         }
301         if ((current_instruct & 0xFF) == 0x25) {
302             reg[R_PC] = 0;
303         }
304         VM_TRAP(current_instruct);
305         break;
306         default:
307         VM_RTI(current_instruct);
308         break;
309     }
310 
311     if (current_instruct == 0) {
312         // END
313         // TODO: add more detailed judge information
314         std::cout << std::endl << "The program ends." << std::endl;
315         return 0;
```

```
316     }
317     return reg[R_PC];
318 }
319
320 } // namespace virtual_machine_nsp
```

程序运行

初次运行应 install symlinks to '/usr/local/bin'

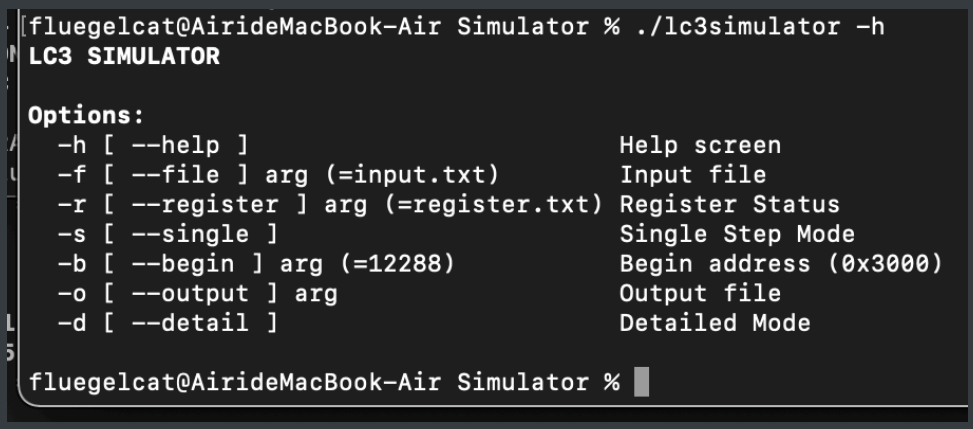
```
1   sudo "/Applications/CMake.app/Contents/bin/cmake-gui" --install
```

在CMakeList.txt所在目录下在终端执行：

```
1 cmake .
2 make
```

这样就生成了makefile文件。

在终端运行：



汇编码：

```
.ORIG x3000
  LD R0, N1
  LD R1, N2
LOOP  ADD R7, R7, R1
  ADD R0, R0, #-1
  BRnp LOOP
  ST R7, N3
  TRAP x25
N1 .FILL #8
N2 .FILL #9
N3 .FILL #0
```

该程序计算先将N1，N2中的值分别存入R0，R1，再计算R0 * R1存入R7。

使用assembler转成机器码存入input.txt内文件如下：

```
0010000000000110
0010001000000110
0001111111000001
0001000000111111
0000101111111101
0011111000000011
1111000000100101
0000000000001000
0000000000001001
0000000000000000
```

```
Last login: Wed Dec 22 16:06:53 on ttys005
fluegelcat@AirideMacBook-Air Simulator % ./lc3simulator -d -f input.txt -o output.txt
LD
R0 = 8, R1 = 0, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 0
COND[NZP] = 001
PC = 3001

LD
R0 = 8, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 0
COND[NZP] = 001
PC = 3002

ADD
R0 = 8, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 9
COND[NZP] = 001
PC = 3003

ADD
R0 = 7, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 9
COND[NZP] = 001
PC = 3004

BR
3005
fffd
R0 = 7, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 9
COND[NZP] = 001
PC = 3002

ADD
R0 = 7, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 12
COND[NZP] = 001
PC = 3003

ADD
R0 = 6, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 12
COND[NZP] = 001
```

最终结果如下：

```
fffd
R0 = 2, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 36
COND[NZP] = 001
PC = 3002

ADD
R0 = 2, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 3f
COND[NZP] = 001
PC = 3003

ADD
R0 = 1, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 3f
COND[NZP] = 001
PC = 3004

BR
3005
fffd
R0 = 1, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 3f
COND[NZP] = 001
PC = 3002

ADD
R0 = 1, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 48
COND[NZP] = 001
PC = 3003

ADD
R0 = 0, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 48
COND[NZP] = 010
PC = 3004

BR
3005
fffd
R0 = 0, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 48
COND[NZP] = 010
PC = 3005

ST
R0 = 0, R1 = 9, R2 = 0, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 48
COND[NZP] = 010
PC = 3006

TRAP
```

R7中存入x0048 PC为x3006 结果正确。