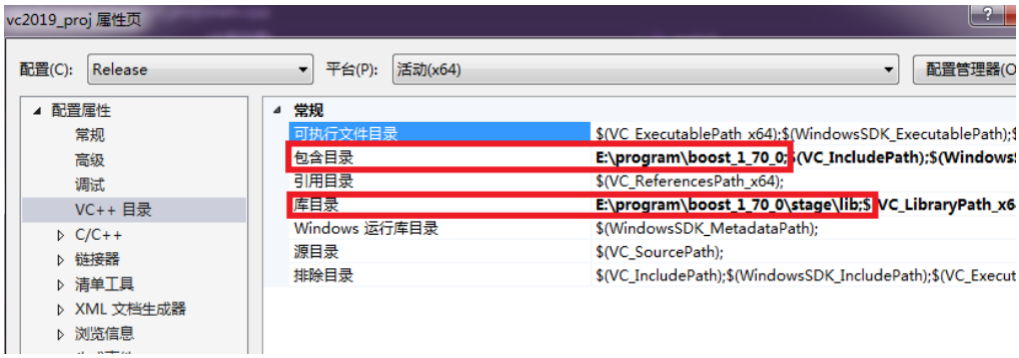
LabS 实验报告

环境配置

1. Cmake：通过助教给出的链接下载压缩包，解压后添加到环境路径
2. 安装 Boost：在官网下载 Boost，在命令行运行 bootstrap.bat，之后运行 bjam.exe 和 b2.exe
3. 在助教给出的框架代码文件夹中新建名为 build 的文件夹，在其中使用命令行输入 cmake..
4. 打开生成好的 vs 文件，进入后配置 boost 环境，如下图：



代码补全

common.h

- 加入头文件：#include

main.cpp

- 需要补全的只有一处，那就是调用执行一条指令的函数，然后计数指令数加一，如下：

```
while(halt_flag) {
        // Single step
        // TO BE DONEd
        virtual_machine.NextStep();
        if (gIsDetailedMode)
            std::cout << virtual_machine.reg << std::endl;
        ++time_flag;
        if (virtual_machine.reg[R_PC] == 0) halt_flag = false;
        if (gIsSingleStepMode) system("pause");
    }
```

memory.cpp

- 第一个函数，从文件中读取机器码，并转化为 16 位的整型数字，存储到 memory 数组中。首先从文件中按行读取字符串，然后调用转化函数，将其转化为 16 位的整型数字，代码如下：

```
void memory_tp::ReadMemoryFromFile(std::string filename, int
beginning_address) {
        // Read from the file
        // TO BE DONE
        std::ifstream input_file;
        input_file.open(filename);
        if (input_file.is_open()) {
            int line_count = std::count(std::istreambuf_iterator<char>
(input_file), std::istreambuf_iterator<char>(), '\n');
```

```
            input_file.close();
            input_file.open(filename);
            std::string tmp;
            while (line_count>=0) {
                input_file >> tmp;
                memory[beginning_address] = TranslateInstruction(tmp);
                beginning_address++;
                line_count--;
            }
            input_file.close();
        }
    }
```

- 第二个和第三个函数则要实现从 memory 数组中根据地址获取内存值，代码如下：

```
    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];
    }
```

simulator

- 首先是位扩展函数：获得了原数，以及原数的位数，需要对其进行符号位扩展。如果原数最高位为 0，那么扩展过程也是补零，而因为原数的类型是 16 位的整型数字，默认它的高位也都是零，所以无需操作。如果原数最高位为 1，那么则需将其转化为对应补码的负数。代码如下：

```
inline T SignExtend(const T x) {
    // Extend the number
    // TO BE DONE
    // DONE
    int16_t tmp_a = 0b1;
    int16_t tmp_result = 0;
    tmp_a = tmp_a << (B - 1);
    if (x & tmp_a) {
        tmp_result = x - tmp_a - tmp_a;
    }
    else tmp_result = x;
    return tmp_result;
}
```

- 根据寄存器更新状态寄存器，即置 NZP 寄存器，代码如下：

```
void virtual_machine_tp::UpdateCondRegister(int regname) {
    // Update the condition register
    // TO BE DONE
    // DONE
    if (reg[regname] < 0) reg[R_COND] = 0b100;
    else if (reg[regname] == 0) reg[R_COND] = 0b10;
    else reg[R_COND] = 0b1;
}
```

- 之后就是各类指令的具体执行，参考课本的附录，进行相应的操作。涉及到内存，则需要 memory 数组；涉及到寄存器，则取相应的寄存器；涉及位扩展，则调用刚写完的函数；涉及 PC，则取 PC。如果发生向寄存器写入值的操作，则需要更新状态寄存器
- SetReg 函数：单步执行函数，根据当前一条指令，进行解码，然后调用相应的指令函数进行操作。

实验总结

- 本次实验的代码部分相比于 LabA 而言比较简单，但是在环境配置方面比较复杂，尤其在 Boost 库的安装与配置上，查阅了很多教程，反复下载了几遍，也尝试了在 Linux 环境下运行，最终还是一点一点分析 boost 的下载提示信息，结合网上找到的说明文档，实现配置。
- 本次实验之所以代码写起来容易，很大程度上是因为在助教写出的框架中，已经把一个大任务拆分成了许多相互配合的小任务，从而大大简化了每个子任务的难度。这也教会了我，在写较大规模的代码时，不要边写边想，而是先建立整体的运行框架、运行逻辑，然后根据思路将其进行拆分。