Simulation lab

姓名：杨涛 学号：PB20020599

配置环境

本lab所用

```
$ sudo apt install cmake
```

填充代码

main

main函数一步的操作，若 `gIsSingleStepMode` 为 `true` ，则对应仿照gdb的交互，当用户输入n时下一步并打印寄存器状态

否则直到最后再打印寄存器状态

```
while (halt_flag)
   {
       // Single step
       char c;
       if (gIsSingleStepMode)
       {

           while (1)
           {
               std::cout << "$ ";
               std::cin >> c;
               if (c == 'n')
               {
                   if (!virtual_machine.NextStep())
                       halt_flag = 0;
                   break;
               }
               else if (c == 'r')
               {
                   gIsSingleStepMode = false;
                   if (!virtual_machine.NextStep())
                       halt_flag = 0;
                   break;
               }
               else
               {
                   std::cout << "No Defined Operation" << std::endl;
               }
           }
       }
       else if (!virtual_machine.NextStep())
           halt_flag = 0;
       if (gIsDetailedMode)
           std::cout << virtual_machine.reg << std::endl;
       ++time_flag;
   }
```

memory

实现memory.h所声明的方法

```
void memory_tp::ReadMemoryFromFile(std::string filename, int beginning_address)
    {
        // Read from the file
        std::ifstream input_file;
        input_file.open(filename);
        std::string line;
        int i = beginning_address;
        if (!input_file.is_open())
        {
            std::cout << "open file error." << std::endl;
            exit(-1);
        }
        while (!input_file.eof())
        {
            getline(input_file, line);
            if (line.length() == 16 || line.length() == 0)
            {
                memory[i] = TranslateInstruction(line);
                i++;
            }
            else
            {
                std::cout << "input error." << std::endl;
                exit(-1); //输入文件错误
            }
        }
        while (i < kVirtualMachineMemorySize)//将剩余部分填充0
        {
            memory[i] = 0;
            i++;
        }
    }

    int16_t memory_tp::GetContent(int address) const
    {
        // get the content
        return memory[address];
    }

    int16_t &memory_tp::operator[](int address)
    {
        // get the content
        return memory[address];
    }
```

simulator

符号拓展

```
template <typename T, unsigned B>
    inline T SignExtend(const T x)
    {
        // Extend the number
        T t=(x&(1<<(B-1)))?(-1)<<B:0;
        return x|t;
    }
```

更新condition code

```
void virtual_machine_tp::UpdateCondRegister(int regname)
    {
        // Update the condition register
        int16_t temp = reg[regname];
        if (temp > 0)
            reg[R_COND] = 0b001;
        else if (temp < 0)
            reg[R_COND] = 0b100;
        else
            reg[R_COND] = 0b010;
    }
```

不同指令对应的改变状态的函数

```
void virtual_machine_tp::VM_ADD(int16_t inst)
    {
        int flag = inst & 0b100000;
        int dr = (inst >> 9) & 0x7;
        int sr1 = (inst >> 6) & 0x7;
        if (flag)
        {
            // add inst number
            int16_t imm = SignExtend<int16_t, 5>(inst & 0b11111);
            reg[dr] = reg[sr1] + imm;
        }
        else
        {
            // add register
            int sr2 = inst & 0x7;
            reg[dr] = reg[sr1] + reg[sr2];
        }
        // Update condition register
        UpdateCondRegister(dr);
    }

    /* 仿照上面ADD指令的操作即可 */
    void virtual_machine_tp::VM_AND(int16_t inst)
    {
        int flag = inst & 0b100000;
        int dr = (inst >> 9) & 0x7;
        int sr1 = (inst >> 6) & 0x7;
        if (flag)
        {
            int16_t imm = SignExtend<int16_t, 5>(inst & 0b11111);
            reg[dr] = reg[sr1] & imm;
        }
        else
        {
            int sr2 = inst & 0x7;
            reg[dr] = reg[sr1] & reg[sr2];
        }
        // Update condition register
        UpdateCondRegister(dr);
    }

    /* 读取根据condition code 决定是否要跳转 */
    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;
```

```
    }
}

void virtual_machine_tp::VM_JMP(int16_t inst)
{
    int BaseR = (inst >> 6) & 0x7;
    reg[R_PC] = reg[BaseR];
}

/* 先判断是JSR还是JSRR，再分别使用pc_offset或BaseR跳转，并更新R7 */
void virtual_machine_tp::VM_JSR(int16_t inst)
{
    int16_t temp = reg[R_PC];

    if (inst & 0b100000000000)
    {
        int16_t pc_offset = SignExtend<int16_t, 11>(inst & 0x7FF);
        reg[R_PC] += pc_offset;
    }
    else
    {
        int BaseR = (inst >> 6) & 0x7;
        reg[R_PC] = reg[BaseR];
    }
    reg[R_R7] = temp;
}

void virtual_machine_tp::VM_LD(int16_t inst)
{
    int16_t dr = (inst >> 9) & 0x7;
    int16_t pc_offset = SignExtend<int16_t, 9>(inst & 0x1FF);
    reg[dr] = mem[reg[R_PC] + pc_offset];
    UpdateCondRegister(dr);
}

void virtual_machine_tp::VM_LDI(int16_t inst)//仿照LD指令即可
{
    int16_t dr = (inst >> 9) & 0x7;
    int16_t pc_offset = SignExtend<int16_t, 9>(inst & 0x1FF);
    reg[dr] = mem[mem[reg[R_PC] + pc_offset]];
    UpdateCondRegister(dr);
}

void virtual_machine_tp::VM_LDR(int16_t inst)
{
    int16_t dr = (inst >> 9) & 0x7;
    int16_t BaseR = (inst >> 6) & 0x7;
    int16_t offset = SignExtend<int16_t, 6>(inst & 0x3F);
    reg[dr] = mem[reg[BaseR] + offset];
    UpdateCondRegister(dr);
}

void virtual_machine_tp::VM_LEA(int16_t inst)
{
```

```
    int16_t dr = (inst >> 9) & 0x7;
    int16_t pc_offset = SignExtend<int16_t, 9>(inst & 0x1FF);
    reg[dr] = reg[R_PC] + pc_offset;
}

void virtual_machine_tp::VM_NOT(int16_t inst)
{
    int dr = (inst >> 9) & 0x7;
    int sr = (inst >> 6) & 0x7;
    reg[dr] = ~reg[sr];
    // Update condition register
    UpdateCondRegister(dr);
}

void virtual_machine_tp::VM_RTI(int16_t inst)
{
    ; // PASS
}

void virtual_machine_tp::VM_ST(int16_t inst)//仿照LD
{
    int16_t sr = (inst >> 9) & 0x7;
    int16_t pc_offset = SignExtend<int16_t, 9>(inst & 0x1FF);
    mem[reg[R_PC] + pc_offset] = reg[sr];
}

void virtual_machine_tp::VM_STI(int16_t inst)
{
    int16_t sr = (inst >> 9) & 0x7;
    int16_t pc_offset = SignExtend<int16_t, 9>(inst & 0x1FF);
    mem[mem[reg[R_PC] + pc_offset]] = reg[sr];
}

void virtual_machine_tp::VM_STR(int16_t inst)
{
    int16_t sr = (inst >> 9) & 0x7;
    int16_t BaseR = (inst >> 6) & 0x7;
    int16_t offset = SignExtend<int16_t, 6>(inst & 0x3F);
    mem[reg[BaseR] + offset] = reg[sr];
}

void virtual_machine_tp::VM_TRAP(int16_t inst)
{
    int trapnum = inst & 0xFF;
    // if (trapnum == 0x25)
    //     exit(0);
    // TODO: build trap program
}
```