计算机体系结构第二次实验报告——五段流水线模拟器实现

本次实验中，参照教材附带模拟器，我设计了一款支持$ADD, ADDIU, BEQZ, LOAD, STORE$的五段流水线模拟器，支持定向选项、打印性能统计信息、打印指令、打印流水线状态、打印寄存器与存储器内容

代码展示

源代码分为三个文件：

```
// pip.hpp: 定义流水线所使用的数据结构、类
#ifndef PIP_H
#define PIP_H
#include<iostream>
#include<string>
#include<vector>
using namespace std;

class Inst{
    public:
        string name;
        int rs, rt, rd, imm;
        int stage;  // 这里的stage不包括STALL, 指代指令最新到达的阶段
        int issue_cycle = 0;
        Inst(string m_name, int m_rs, int m_rt, int m_td, int m_imm);
};

class status{
    public:
        int Inst_index;
        int stage;
        status(int m_Inst_index, int m_stage);
};

class pipline{
    private:
        vector<Inst> insts;
        int PC = 0;
        bool redirect = 0;
        vector<vector<status>> pip_log;
        int cycle = 1;
        int * reg;
        int * mem;
        int reg_num = 32;
        int mem_num = 1024;
        int n_stall_RAW = 0;
        int n_stall_branch = 0;
        int n_stall_struct = 0;
        int n_branch = 0;
        int branch_taken = 0;
        bool RAW(vector<status>& last, int op_reg, int index, bool redirect);
    public:
        void set_redirect();
        void insts_init(Inst & m_inst);
        void storages_init();
        void storage_set(int index, int value);
        void single_step();
        void multi_step(int n);
        void run_to_breakpoint();
        void run_to_the_end();
        void print_log();
        void print_mem(int end);
        void print_reg(int end);
        void print_performance();
};

#endif
```

> 代码中定义了指令类，包含指令名称（即操作码）、rs、rt、rd、imm等操作数、指令已完成的阶段、指令流出的周期数，还定义了状态类，对应时钟周期图中每个方格中的状态，状态类包含指令的序号（可以理解为指令在内存中的地址）、以及当前状态。对状态的定义使用了int类型，各个数字的含义如下：

数字	含义	数字	含义	数字	含义
0	未流出	1	IF段	2	ID段
3	ID段不成功	4	EX段	5	MEM段
6	WB段	7	执行完成	8	STALL

> 最后定义了pipline类，私有成员包含存储指令的vector向量、PC寄存器（假设指令按1计数）、重定向功能开关、流水线时钟周期图的记录矩阵、时钟周期、寄存器和存储器对印的int数组头指针和大小（寄存器和存储器需要用户手动初始化）、用于统计的量（包含各种stall的类型统计、分支数统计、分支成功数量统计），pipline类的方法除了检测RAW冲突的方法$RAW()$之外都属于公有方法供用户调用，这些方法包括：设置定向功能（默认关闭）、指令初始化、存储空间初始化、内存设置、单步执行、多步执行、执行到断点、执行到结束、打印时钟周期图、打印存储器、打印寄存器、打印统计结果。

```cpp
// pip.cpp: 对应的类方法实现
#include "pip.hpp"

Inst::Inst(string m_name, int m_rs, int m_rt, int m_rd, int m_imm){
    name = m_name;
    rs = m_rs;
    rt = m_rt;
    rd = m_rd;
    imm = m_imm;
    stage = 0;
}

status::status(int m_Inst_index, int m_stage){
    Inst_index = m_Inst_index;
    stage = m_stage;
}
void pipline::set_redirect(){
    redirect = true;
}

// 0: UNISSUE, 1: IF, 2: ID, 3: ID_bad, 4: EX, 5: MEM, 6: WB, 7: FINISH, 8: STALL
void pipline::insts_init(Inst & minst){
    insts.push_back(minst);
    PC = 0;
}

void pipline::storages_init(){
    reg = new int[reg_num];
    mem = new int[mem_num];
    for(int i = 0; i < reg_num; i++)
        reg[i] = 0;
    for(int i = 0; i < mem_num; i++)
        mem[i] = 0;
}

void pipline::memory_set(int index, int value){
    mem[index] = value;
}

// 0: UNISSUE, 1: IF, 2: ID, 3: ID_bad, 4: EX, 5: MEM, 6: WB, 7: FINISH, 8: STALL
// RAW: read after write, 返回true表示有RAW
bool pipline::RAW(vector<status> & last, int op_reg, int index, bool redirect){
    int res = false;
    int barrier = redirect ? 4 : 6; // 若无定向, 则pre_log是WB阶段之前的指令都不能读取;
若有定向, 则pre_log是EX阶段之前的指令不能读取,  3是EX, 5是MEM
    for(int i = index - 1; i >= 0; i--){
        Inst inst = insts[last[i].Inst_index];
        if(inst.name == "ADDU" && inst.rt == op_reg && inst.stage < barrier ||
           inst.name == "LW"   && inst.rt == op_reg && inst.stage < barrier ||
           inst.name == "ADD"  && inst.rd == op_reg && inst.stage < barrier){
            res = true;
```

```
                break;
            }
        }
    return res;
}
// 0: UNISSUE, 1: IF, 2: ID, 3: ID_bad, 4: EX, 5: MEM, 6: WB, 7: FINISH, 8: STALL
void pipline::single_step(){
    vector<bool> occupied(5, false); // 0: IF, 1: ID, 2: EX, 3: MEM, 4: WB
    vector<status> new_log;
    bool stall = false;
    bool new_inst = true;
    if(!pip_log.empty()){
        auto last = pip_log.back();
        for(int i = 0; i < last.size(); i++){
            int inst_stage = insts[last[i].Inst_index].stage;
            if(stall){ // 如果前面有指令stall, 那么后面的指令都要stall
                new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                continue;
            }
            if(inst_stage == 1){ // IF -> ID
                if(occupied[1]){
                    new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                    stall = true;
                    n_stall_struct++;
                }
                else{
                    Inst inst = insts[last[i].Inst_index];
                    if(inst.name == "ADDU"){
                        int op_reg = inst.rs;
                        bool raw = RAW(last, op_reg, i, redirect);
                        if(raw){
                            new_log.push_back(status(last[i].Inst_index, 3)); // ID_bad
                            insts[last[i].Inst_index].stage = 3;
                            n_stall_RAW++;
                        }
                        else{
                            new_log.push_back(status(last[i].Inst_index, 2)); // ID
                            insts[last[i].Inst_index].stage = 2;
                        }
                        occupied[1] = true;
                    }
                    if(inst.name == "ADD"){
                        int op_reg1 = inst.rs;
                        int op_reg2 = inst.rt;
                        bool raw1 = RAW(last, op_reg1, i, redirect), raw2 = RAW(last,
op_reg2, i, redirect);
                        if(raw1 || raw2){
                            new_log.push_back(status(last[i].Inst_index, 3)); // ID_bad
                            insts[last[i].Inst_index].stage = 3;
                            n_stall_RAW++;
                        }
                        else{
                            new_log.push_back(status(last[i].Inst_index, 2)); // ID
                            insts[last[i].Inst_index].stage = 2;
                        }
                        occupied[1] = true;
                    }
                    if(inst.name == "LW"){
                        int op_reg = inst.rs;
                        bool raw = RAW(last, op_reg, i, redirect);
                        if(raw){
                            new_log.push_back(status(last[i].Inst_index, 3)); // ID_bad
                            insts[last[i].Inst_index].stage = 3;
                            n_stall_RAW++;
                        }
                        else{
                            new_log.push_back(status(last[i].Inst_index, 2)); // ID
                            insts[last[i].Inst_index].stage = 2;
                        }
                        occupied[1] = true;
                    }
                    if(inst.name == "BEQZ"){
                        int op_reg = inst.rs;
                        bool raw = RAW(last, op_reg, i, redirect);
                        if(raw){
                            new_log.push_back(status(last[i].Inst_index, 3)); // ID_bad
                            insts[last[i].Inst_index].stage = 3;
                            n_stall_RAW++;
                        }
                        else{
                            new_log.push_back(status(last[i].Inst_index, 2)); // ID
                            insts[last[i].Inst_index].stage = 2;
                            if(reg[inst.rs] == 0){
                                PC += inst.imm;
                                branch_taken++;
                            }
                            new_inst = false;
                            n_stall_branch++;
                        }
                        occupied[1] = true;
                    }
                    if(inst.name == "SW"){
                        int op_reg1 = inst.rs;
                        int op_reg2 = inst.rt;
                        bool raw1 = RAW(last, op_reg1, i, redirect), raw2 = RAW(last,
op_reg2, i, redirect);
                        if(raw1 || raw2){
                            new_log.push_back(status(last[i].Inst_index, 3)); // ID_bad
                            insts[last[i].Inst_index].stage = 3;
                            n_stall_RAW++;
                        }
                        else{
                            new_log.push_back(status(last[i].Inst_index, 2)); // ID
                            insts[last[i].Inst_index].stage = 2;
                        }
                        occupied[1] = true;
                    }
                }
            }
            else if(inst_stage == 2){ // ID
                if(occupied[2]){
                    new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                    stall = true;
                    n_stall_struct++;
                }
                else{
                    new_log.push_back(status(last[i].Inst_index, 4)); // EX
                    insts[last[i].Inst_index].stage = 4;
                    occupied[2] = true;
                }
            }
            else if(inst_stage == 3){ // ID_bad 尝试ID，若无法ID，则STALL
                Inst inst = insts[last[i].Inst_index];
                if(inst.name == "ADDU"){
                    int op_reg = inst.rs;
                    bool raw = RAW(last, op_reg, i, redirect);
                    if(raw){
                        new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                        stall = true;
                        n_stall_RAW++;
                    }
                    else if(!occupied[1]){
                        new_log.push_back(status(last[i].Inst_index, 2)); // ID
                        insts[last[i].Inst_index].stage = 2;
                        occupied[1] = true;
                    }
                    else{
                        new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                        stall = true;
                        n_stall_struct++;
                    }
                }
                if(inst.name == "ADD"){
                    int op_reg1 = inst.rs;
                    int op_reg2 = inst.rt;
                    bool raw1 = RAW(last, op_reg1, i, redirect), raw2 = RAW(last,
op_reg2, i, redirect);
                    if(raw1 || raw2){
                        new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                        stall = true;
                        n_stall_RAW++;
                    }
                    else if(!occupied[1]){
                        new_log.push_back(status(last[i].Inst_index, 2)); // ID
                        insts[last[i].Inst_index].stage = 2;
                        occupied[1] = true;
                    }
                    else{
                        new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                        stall = true;
                        n_stall_struct++;
                    }
                }
                if(inst.name == "LW"){
                    int op_reg = inst.rs;
                    bool raw = RAW(last, op_reg, i, redirect);
                    if(raw){
                        new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                        stall = true;
                        n_stall_RAW++;
                    }
                    else if(!occupied[1]){
                        new_log.push_back(status(last[i].Inst_index, 2)); // ID
                        insts[last[i].Inst_index].stage = 2;
                        occupied[1] = true;
                    }
                    else{
                        new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                        stall = true;
                        n_stall_struct++;
                    }
                }
                if(inst.name == "BEQZ"){
                    int op_reg = inst.rs;
                    bool raw = RAW(last, op_reg, i, redirect);
                    if(raw){
                        new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                        stall = true;
                        n_stall_RAW++;
                    }
                    else if(!occupied[1]){
                        new_log.push_back(status(last[i].Inst_index, 2)); // ID
                        insts[last[i].Inst_index].stage = 2;
                        if(reg[inst.rs] == 0){
                            PC += inst.imm;
                            branch_taken++;
                        }
                        new_inst = false;
                        n_stall_branch++;
                        occupied[1] = true;
                    }
                    else{
                        new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                        stall = true;
                        n_stall_struct++;
                    }
                }
                if(inst.name == "SW"){
                    int op_reg1 = inst.rs;
                    int op_reg2 = inst.rt;
                    bool raw1 = RAW(last, op_reg1, i, redirect), raw2 = RAW(last,
op_reg2, i, redirect);
                    if(raw1 || raw2){
                        new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                        stall = true;
                        n_stall_RAW++;
                    }
                    else if(!occupied[1]){
                        new_log.push_back(status(last[i].Inst_index, 2)); // ID
                        insts[last[i].Inst_index].stage = 2;
                        occupied[1] = true;
                    }
                    else{
                        new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                        stall = true;
                        n_stall_struct++;
                    }
                }
            }
            else if(inst_stage == 4){ // EX
                if(occupied[3]){
                    new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                    stall = true;
                    n_stall_struct++;
                }
                else{
                    Inst inst = insts[last[i].Inst_index];
                    if(inst.name == "SW"){
                        int addr = reg[inst.rs] + inst.imm;
                        mem[addr] = reg[inst.rt];
                    }
                    new_log.push_back(status(last[i].Inst_index, 5)); // MEM
                    insts[last[i].Inst_index].stage = 5;
                    occupied[3] = true;
                }
            }
            else if(inst_stage == 5){ // MEM
                if(occupied[4]){
                    new_log.push_back(status(last[i].Inst_index, 8)); // STALL
                    stall = true;
                    n_stall_struct++;
                }
                else{
                    Inst inst = insts[last[i].Inst_index];
                    if(inst.name == "ADDU"){
                        reg[inst.rt] = reg[inst.rs] + inst.imm;
                    }
                    if(inst.name == "ADD"){
                        reg[inst.rd] = reg[inst.rs] + reg[inst.rt];
                    }
                    if(inst.name == "LW"){
                        int addr = reg[inst.rs] + inst.imm;
                        reg[inst.rt] = mem[addr];
                    }
                    new_log.push_back(status(last[i].Inst_index, 6)); // WB
                    insts[last[i].Inst_index].stage = 6;
                    occupied[4] = true;
                }
            }
            else if(inst_stage == 6){ // WB
                insts[last[i].Inst_index].stage = 7; // FINISH
                if(PC == insts.size() && last[i].Inst_index == PC - 1){
                    return;
                }
            }
        }
    }
    // issue one new instruction
    if(!stall && new_inst && PC < insts.size()){
        new_log.push_back(status(PC, 1));
        if(insts[PC].name == "BEQZ"){
            n_branch++;
        }
        insts[PC].stage = 1;
        insts[PC].issue_cycle = cycle;
        PC++;
    }
    pip_log.push_back(new_log);
    cycle++;
}
// 0: UNISSUE, 1: IF, 2: ID, 3: ID_bad, 4: EX, 5: MEM, 6: WB, 7: FINISH, 8: STALL

void pipline::multi_step(int n){
    while(n--){
        single_step();
    }
}

void pipline::run_to_the_end(){
    while(insts.back().stage != 6){
        single_step();
    }
}

void pipline::run_to_breakpoint(){
    int index;
    string stage;
    printf("请插入断点\n");
    printf("请输入断点指令序号和阶段\n");
    printf("指令序号: ");
    while(true){
        scanf("%d", &index);
        if(index >= insts.size()){
            printf("指令序号超出范围, 请重新输入\n");
            printf("指令序号: ");
            continue;
        }
        break;
    }
    printf("阶段: ");
    cin >> stage;
    int stage_num;
    while(true){
        if(stage == "IF")
            stage_num = 1;
        else if(stage == "ID")
            stage_num = 2;
        else if(stage == "EX")
            stage_num = 4;
        else if(stage == "MEM")
            stage_num = 5;
        else if(stage == "WB")
            stage_num = 6;
        else {
            printf("阶段输入错误, 请重新输入\n");
            printf("阶段: ");
            cin >> stage;
            continue;
        }
        break;
    }
    while(insts[index].stage < stage_num){
        single_step();
        if(insts[index].stage >= stage_num)
            break;
    }
}

void pipline::print_reg(int end){
    for(int i = 0; i < end; i+=8){
        for(int j = 0; j < 8; j++){
            printf("REG%7d  ", i + j);
        }
        printf("\n");
        for(int j = 0; j < 8; j++){
            printf("0x%08x  ", reg[i + j]);
        }
        printf("\n\n");
    }
}

void pipline::print_mem(int end){
    for(int i = 0; i < end; i+=8){
        printf("MEM%-4d ", i);
        for(int j = 0; j < 8; j++){
            printf("0x%08x  ", mem[i + j]);
        }
        printf("\n\n");
    }
}

void pipline::print_log(){
    int n_inst = insts.size();
    for(int i = 0; i < n_inst; i++){
        string name = insts[i].name;
        if(name == "ADDU")
            printf("Inst %d:    %4s $%d, $%d, %d", i, insts[i].name.c_str(),
insts[i].rt, insts[i].rs, insts[i].imm);
        else if(name == "LW" || name == "SW")
            printf("Inst %d:    %4s $%d, %d($%d)", i, insts[i].name.c_str(),
insts[i].rt, insts[i].imm, insts[i].rs);
        else if(name == "BEQZ")
            printf("Inst %d:    %4s $%d, %d", i, insts[i].name.c_str(), insts[i].rs,
insts[i].imm);
        else if(name == "ADD")
            printf("Inst %d:    %4s $%d, $%d, $%d", i, insts[i].name.c_str(),
insts[i].rd, insts[i].rs, insts[i].rt);
        printf("\n");
    }
    printf("-------------------------------\n");
    for(int i = 0; i < n_inst; i++){
        printf("Inst %d:", i);
        int n_space = insts[i].issue_cycle - 1;
        n_space = max(n_space, 0);
        for(int j = 0; j < n_space; j++)
            printf("      ");
        for(auto log : pip_log){
            for(auto s : log){
                if(s.Inst_index == i){
                    int stage = s.stage;
                    if(stage == 1)
                        printf(" %5s", "IF");
                    else if(stage == 2)
                        printf(" %5s", "ID");
                    else if(stage == 3)
                        printf(" %5s", "ID*");
                    else if(stage == 4)
                        printf(" %5s", "EX");
                else if(stage == 5)
                    printf(" %5s", "MEM");
                else if(stage == 6)
                    printf(" %5s", "WB");
                else if(stage == 8)
                    printf(" %5s", "STALL");
                break;
            }
        }
    }
    printf("\n");
  }
}

void pipline::print_performance(){
    // 周期数、定向机制、RAW停顿、控制停顿、分支指令条数、成功与失败占比、load store指令条数、占比
    printf("Cycle_index: %d\n", cycle - 1);
    printf("Redirection: ");
    if(redirect) printf("ON");
    else printf("OFF");
    printf("\n");
    printf("RAW_stall: %d\n", n_stall_RAW);
    printf("Control_stall: %d\n", n_stall_branch);
    printf("Struction_stall: %d\n", n_stall_struct);
    printf("Branch_inst: %d\n", n_branch);
    printf("Branch_success_rate: %f\n", (float)(branch_taken) / n_branch);
}
```

> 以上代码包含头文件中提到的所有内容的详细实现，其中最重要的是单步执行功能的实现，提取了上一个时钟周期的执行记录，并根据该记录推进本周期的执行，还要检测是否还有可以执行的指令。

```
// main.cpp: 调用上面两个文件中函数的代码，只需要在文件头加上#include "pip.hpp"即可，需要将pip头文件或二进制可链接文件
// 放在同一目录下
#include "pip.hpp"
using namespace std;
int main(){
    vector<Inst> insts;
    // addu rt, rs, imm          read rs,      write rt
    // add  rd, rs, rt           read rs, rt   write rd
    // lw   rt, rs, imm          read rs,      write rt
    // sw   rt, rs, imm          read rs, rt
    // beqz rs, imm              read rs

    // addu r1, r0, 32
    // lw   r3, r1, 0
    // lw   r4, r1, 0
    // beqz r5, 0x14
    // addu r3, r3, 1111
    // sw   r3, r1, 0
    insts.push_back(Inst("ADDU", 0, 1, 0, 32));      // addu r1, r0, 32
    insts.push_back(Inst("LW"  , 1, 3, 0, 0));       // lw   r3, r1, 0
    insts.push_back(Inst("LW"  , 1, 4, 0, 0));       // lw   r4, r1, 0
    insts.push_back(Inst("BEQZ", 5, 0, 0, 0));       // beqz r5, 0(PC)
    insts.push_back(Inst("ADDU", 3, 3, 0, 1111));    // addu r3, r3, 1111
    insts.push_back(Inst("SW"  , 1, 3, 0, 0));       // sw   r3, r1, 0
    pipline p;
    p.storages_init();
    for(int i = 0; i < insts.size(); i++){
        p.insts_init(insts[i]);
    }
    p.memory_set(32, 3172);

    // p.set_redirect();
    // p.run_to_the_end();
    // p.single_step();
    p.multi_step(16);
    p.print_log();
    // p.print_mem(64);
    // p.print_reg(16);
    puts("");
    p.print_performance();
    return 0;
}
```

> main文件中应用了定义的类和它的方法，流水线的实现到此完成。下面是编译部分的演示：

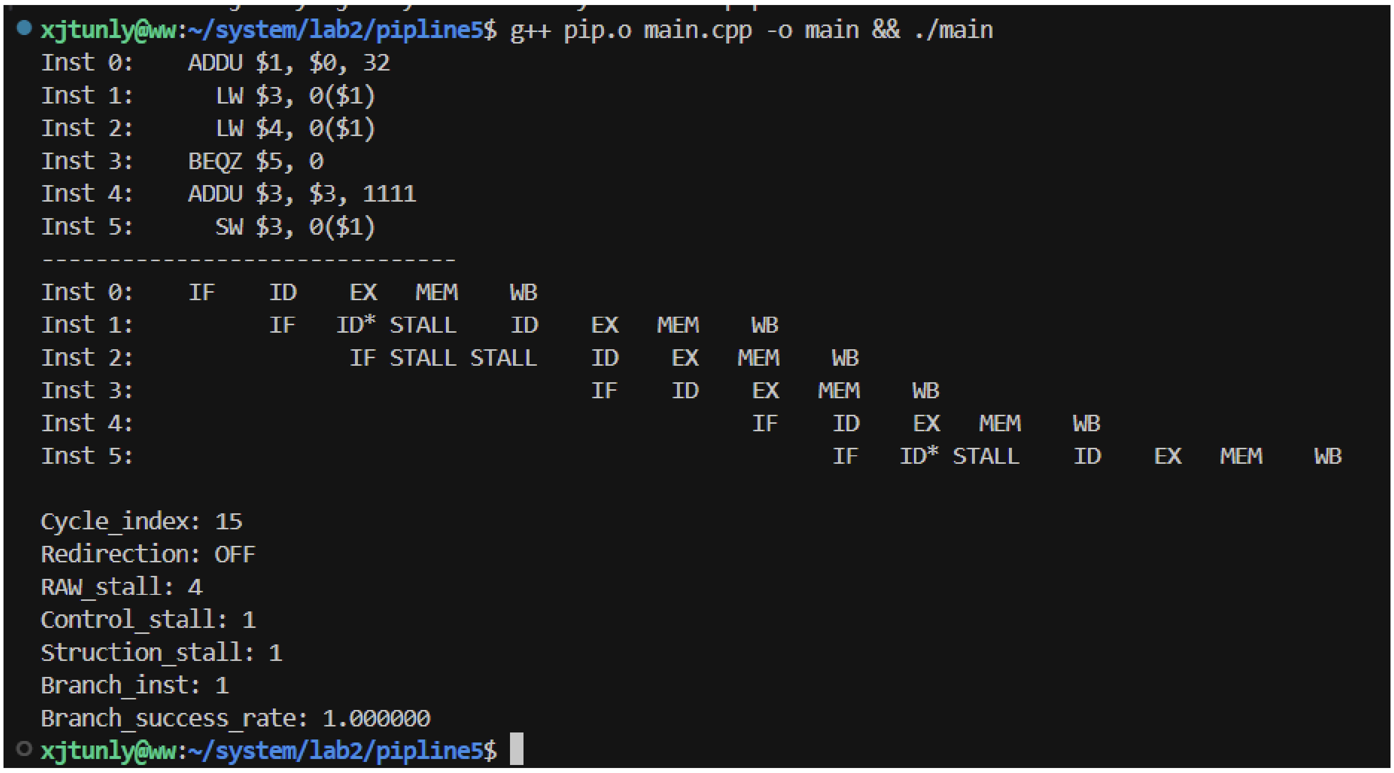
编译演示

所编写的文件中不包含与操作系统的系统调用强相关的语句，所以可以跨平台编译，我使用了Ubuntu 22.04 + gcc 11.4.0编译该文件。

编译运行指令：

```
$ g++ -c pip.hpp pip.cpp
$ g++ pip.o main.cpp -o main
$ ./main
```

使用上述的main.cpp文件，得到的输出为：



```
xjtunly@ww:~/system/lab2/pipline5$ g++ pip.o main.cpp -o main && ./main
Inst 0:    ADDU $1, $0, 32
Inst 1:      LW $3, 0($1)
Inst 2:      LW $4, 0($1)
Inst 3:    BEQZ $5, 0
Inst 4:    ADDU $3, $3, 1111
Inst 5:      SW $3, 0($1)
-------------------------------
Inst 0:    IF    ID    EX   MEM    WB
Inst 1:          IF   ID* STALL    ID    EX   MEM    WB
Inst 2:                IF STALL STALL    ID    EX   MEM    WB
Inst 3:                                  IF    ID    EX   MEM    WB
Inst 4:                                              IF    ID    EX   MEM    WB
Inst 5:                                                    IF   ID* STALL    ID    EX   MEM    WB

Cycle_index: 15
Redirection: OFF
RAW_stall: 4
Control_stall: 1
Struction_stall: 1
Branch_inst: 1
Branch_success_rate: 1.000000
xjtunly@ww:~/system/lab2/pipline5$ 
```

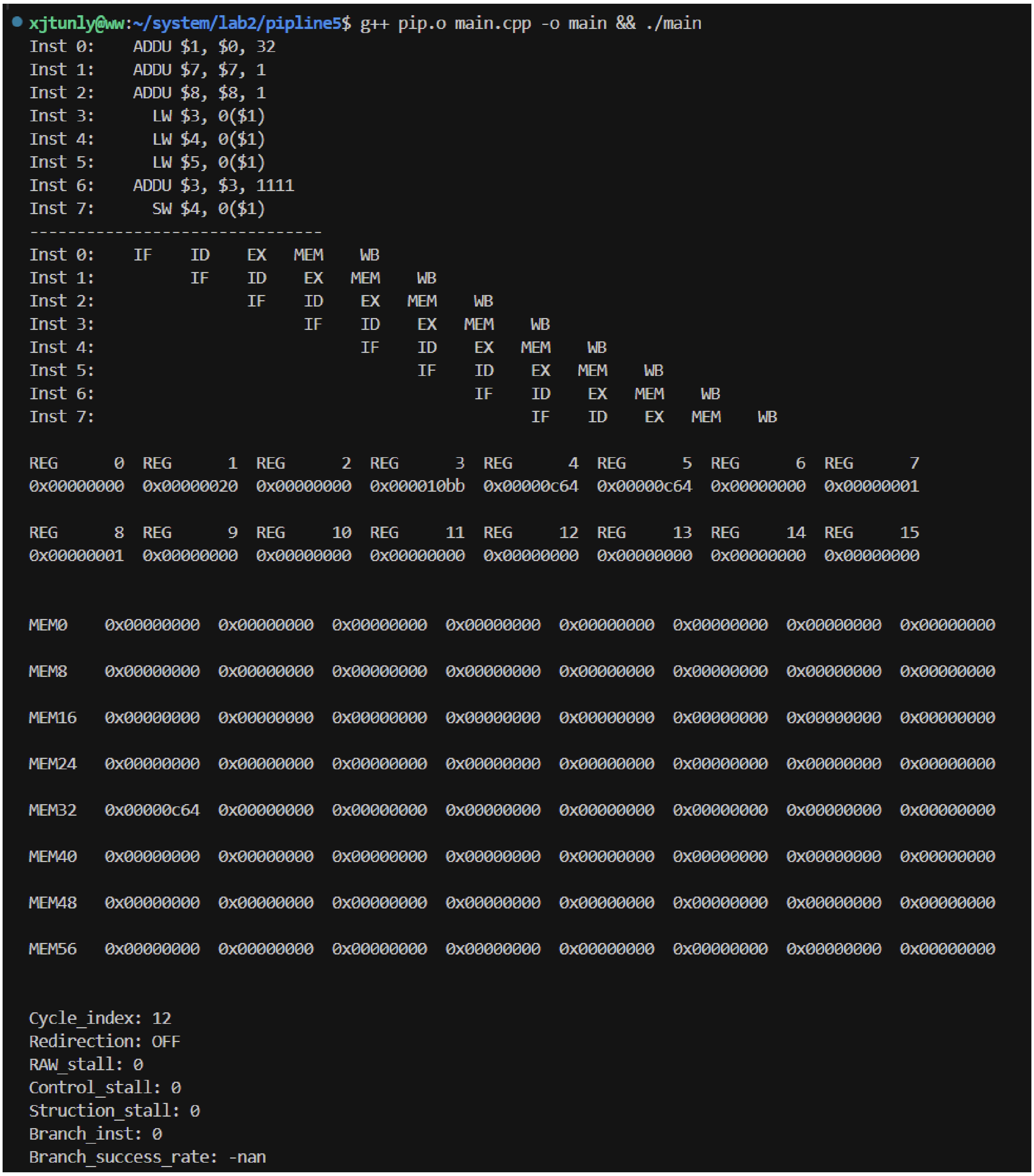
可见程序运行正常。

实验任务一：无冲突的流水线场景

> 为了对比起见，使用了第一次实验中的代码。对main.cpp修改得到：

```cpp
#include "pip.hpp"
using namespace std;
void model1();
int main(){
    // addu rt, rs, imm          read rs,      write rt
    // add  rd, rs, rt           read rs, rt   write rd
    // lw   rt, rs, imm          read rs,      write rt
    // sw   rt, rs, imm          read rs, rt
    // beqz rs, imm              read rs
    model1();
    return 0;
}

void model1(){
    // ADDU  R1, R0, 32
    // ADDU  R7, R7, 1
    // ADDU  R8, R8, 1
    // LW    R3, 0(R1)
    // LW    R4, 0(R1)
    // Lw    R5, 0(R1)
    // ADDU  R3, R3, 1111
    // SW    R4, 0(R1)
    vector<Inst> inst;
    inst.push_back(Inst("ADDU", 0, 1, 0, 32));
    inst.push_back(Inst("ADDU", 7, 7, 0, 1));
    inst.push_back(Inst("ADDU", 8, 8, 0, 1));
    inst.push_back(Inst("LW"  , 1, 3, 0, 0));
    inst.push_back(Inst("LW"  , 1, 4, 0, 0));
    inst.push_back(Inst("LW"  , 1, 5, 0, 0));
    inst.push_back(Inst("ADDU", 3, 3, 0, 1111));
    inst.push_back(Inst("SW"  , 1, 4, 0, 0));
    pipline p;
    p.storages_init();
    for(int i = 0; i < inst.size(); i++){
        p.insts_init(inst[i]);
    }
    p.memory_set(32, 3172);
    p.run_to_the_end();
    p.print_log();
    puts("");
    p.print_reg(16);
    puts("");
    p.print_mem(64);
    puts("");
    p.print_performance();
}
```

执行的输出效果为：



```
xjtunly@ww:~/system/lab2/pipline5$ g++ pip.o main.cpp -o main && ./main
Inst 0:    ADDU $1, $0, 32
Inst 1:    ADDU $7, $7, 1
Inst 2:    ADDU $8, $8, 1
Inst 3:      LW $3, 0($1)
Inst 4:      LW $4, 0($1)
Inst 5:      LW $5, 0($1)
Inst 6:    ADDU $3, $3, 1111
Inst 7:      SW $4, 0($1)
-------------------------------
Inst 0:    IF    ID    EX   MEM    WB
Inst 1:          IF    ID    EX   MEM    WB
Inst 2:                IF    ID    EX   MEM    WB
Inst 3:                      IF    ID    EX   MEM    WB
Inst 4:                            IF    ID    EX   MEM    WB
Inst 5:                                  IF    ID    EX   MEM    WB
Inst 6:                                        IF    ID    EX   MEM    WB
Inst 7:                                              IF    ID    EX   MEM    WB

REG       0  REG       1  REG       2  REG       3  REG       4  REG       5  REG       6  REG       7
0x00000000  0x00000020  0x00000000  0x000010bb  0x00000c64  0x00000c64  0x00000000  0x00000001

REG       8  REG       9  REG      10  REG      11  REG      12  REG      13  REG      14  REG      15
0x00000001  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000


MEM0     0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM8     0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM16    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM24    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM32    0x00000c64  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM40    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM48    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM56    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000


Cycle_index: 12
Redirection: OFF
RAW_stall: 0
Control_stall: 0
Struction_stall: 0
Branch_inst: 0
Branch_success_rate: -nan
```

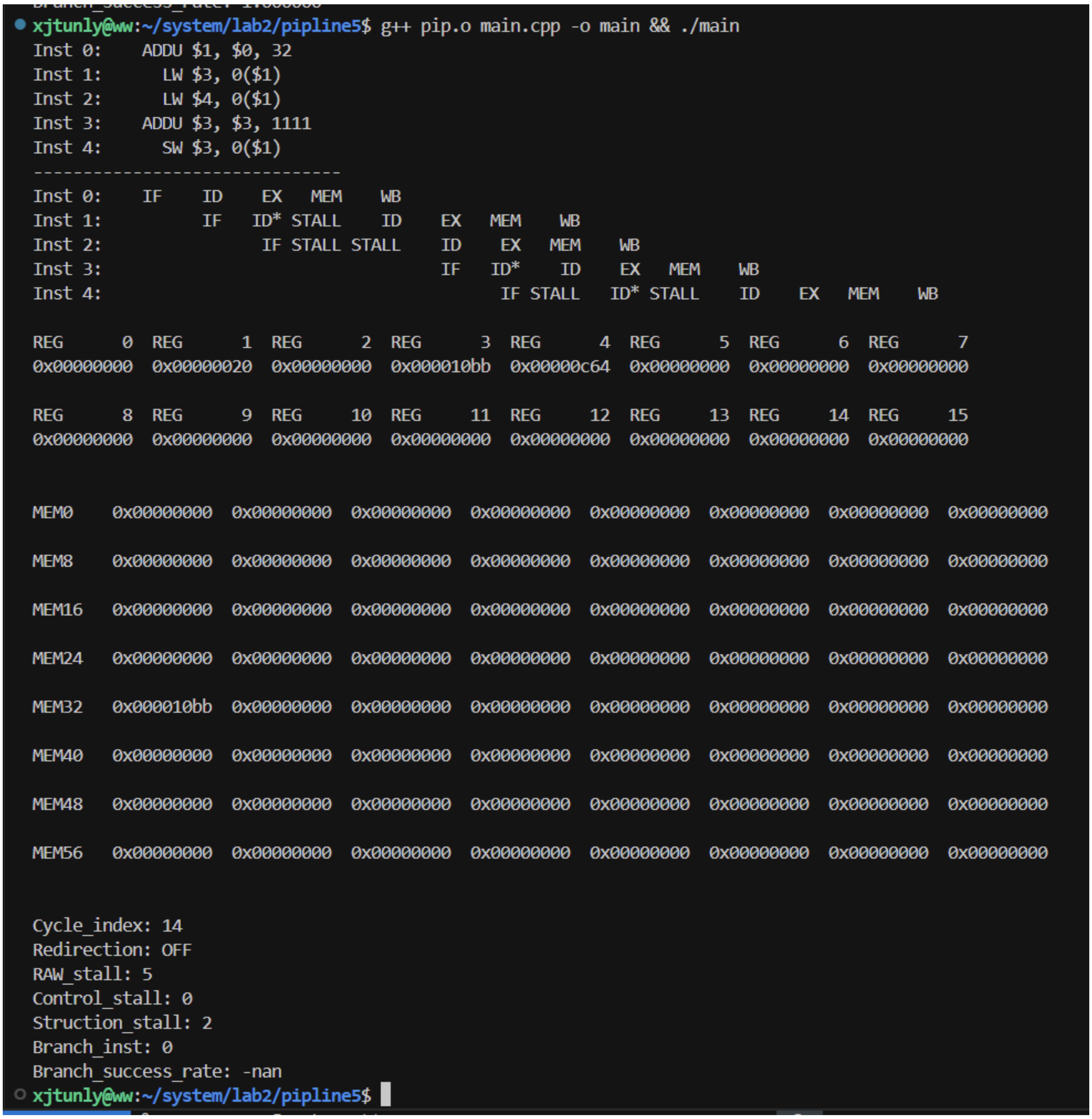
与第一次实验所得结果完全一致。

实验任务二：关闭定向功能，加入RAW相关

> main.cpp和执行结果如下所示

```
#include "pip.hpp"
using namespace std;
```

```cpp
void model1();
void model2();
int main(){
    // addu rt, rs, imm         read rs,      write rt
    // add  rd, rs, rt          read rs, rt   write rd
    // lw   rt, rs, imm         read rs,      write rt
    // sw   rt, rs, imm         read rs, rt
    // beqz rs, imm             read rs
    model2();
    return 0;
}
void model2(){
    // ADDU  R1, R0, 32
    // LW    R3, 0(R1)
    // LW    R4, 0(R1)
    // ADDU  R3, R3, 1111
    // SW    R3, 0(R1)
    vector<Inst> inst;
    inst.push_back(Inst("ADDU", 0, 1, 0, 32));
    inst.push_back(Inst("LW"  , 1, 3, 0, 0));
    inst.push_back(Inst("LW"  , 1, 4, 0, 0));
    inst.push_back(Inst("ADDU", 3, 3, 0, 1111));
    inst.push_back(Inst("SW"  , 1, 3, 0, 0));
    pipline p;
    p.storages_init();
    for(int i = 0; i < inst.size(); i++){
        p.insts_init(inst[i]);
    }
    p.memory_set(32, 3172);
    p.run_to_the_end();
    p.print_log();
    puts("");
    p.print_reg(16);
    puts("");
    p.print_mem(64);
    puts("");
    p.print_performance();
}
```



```
xjtunly@ww:~/system/lab2/pipline5$ g++ pip.o main.cpp -o main && ./main
Inst 0:    ADDU $1, $0, 32
Inst 1:      LW $3, 0($1)
Inst 2:      LW $4, 0($1)
Inst 3:    ADDU $3, $3, 1111
Inst 4:      SW $3, 0($1)
-------------------------------
Inst 0:    IF    ID    EX   MEM    WB
Inst 1:          IF   ID* STALL    ID    EX   MEM    WB
Inst 2:                IF STALL STALL    ID    EX   MEM    WB
Inst 3:                                  IF   ID*    ID    EX   MEM    WB
Inst 4:                                        IF STALL   ID* STALL    ID    EX   MEM    WB

REG      0  REG      1  REG      2  REG      3  REG      4  REG      5  REG      6  REG      7
0x00000000  0x00000020  0x00000000  0x000010bb  0x00000c64  0x00000000  0x00000000  0x00000000

REG      8  REG      9  REG     10  REG     11  REG     12  REG     13  REG     14  REG     15
0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000


MEM0    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM8    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM16   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM24   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM32   0x000010bb  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM40   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM48   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM56   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000


Cycle_index: 14
Redirection: OFF
RAW_stall: 5
Control_stall: 0
Struction_stall: 2
Branch_inst: 0
Branch_success_rate: -nan
xjtunly@ww:~/system/lab2/pipline5$
```

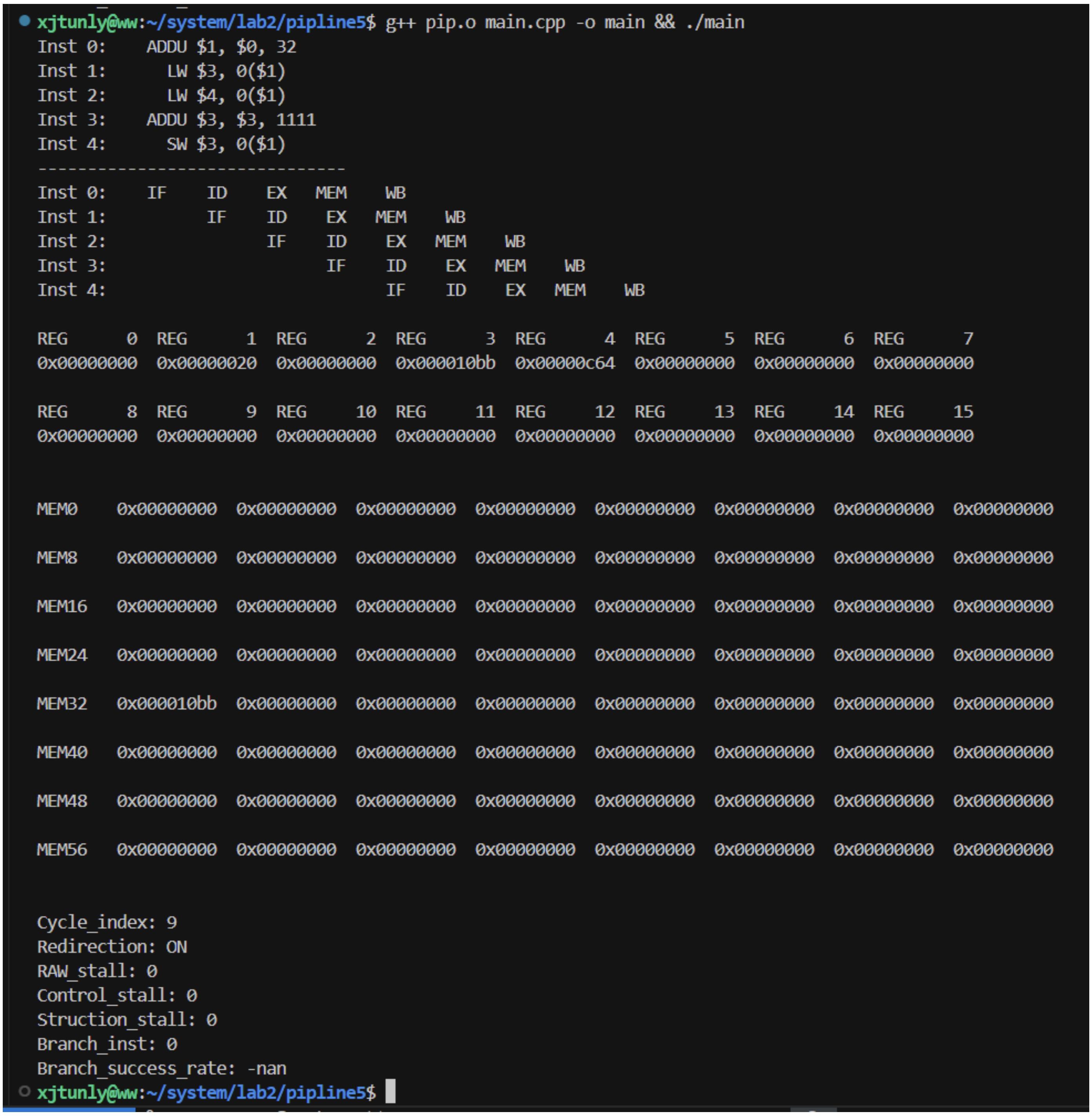
与第一次实验结果完全一致。

实验任务三：开启定向功能，加入RAW相关

修改main.cpp，代码与执行结果如下：

```
#include "pip.hpp"
using namespace std;
void model3();
int main(){
    // addu rt, rs, imm          read rs,      write rt
    // add  rd, rs, rt           read rs, rt   write rd
```

```cpp
    // lw   rt, rs, imm          read rs,      write rt
    // sw   rt, rs, imm          read rs, rt
    // beqz rs, imm              read rs
    model3();
    return 0;
}
void model3(){
    // ADDU  R1, R0, 32
    // LW    R3, 0(R1)
    // LW    R4, 0(R1)
    // ADDU  R3, R3, 1111
    // SW    R3, 0(R1)
    vector<Inst> inst;
    inst.push_back(Inst("ADDU", 0, 1, 0, 32));
    inst.push_back(Inst("LW"  , 1, 3, 0, 0));
    inst.push_back(Inst("LW"  , 1, 4, 0, 0));
    inst.push_back(Inst("ADDU", 3, 3, 0, 1111));
    inst.push_back(Inst("SW"  , 1, 3, 0, 0));
    pipline p;
    p.storages_init();
    p.set_redirect();
    for(int i = 0; i < inst.size(); i++){
        p.insts_init(inst[i]);
    }
    p.memory_set(32, 3172);
    p.run_to_the_end();
    p.print_log();
    puts("");
    p.print_reg(16);
    puts("");
    p.print_mem(64);
    puts("");
    p.print_performance();
}
```

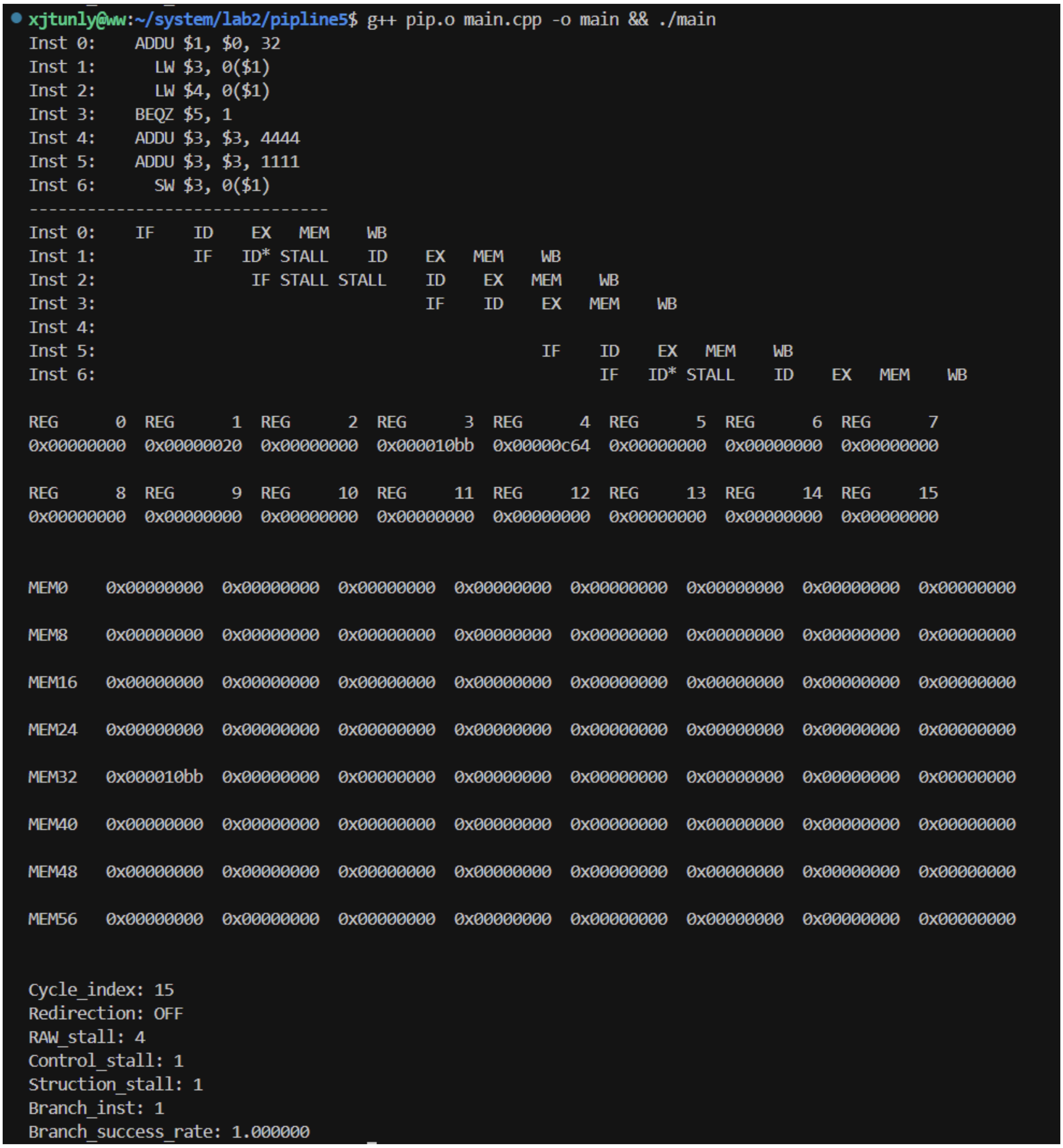

```
● xjtunly@ww:~/system/lab2/pipline5$ g++ pip.o main.cpp -o main && ./main
  Inst 0:    ADDU $1, $0, 32
  Inst 1:      LW $3, 0($1)
  Inst 2:      LW $4, 0($1)
  Inst 3:    ADDU $3, $3, 1111
  Inst 4:      SW $3, 0($1)
  ------------------------------
  Inst 0:    IF    ID    EX   MEM    WB
  Inst 1:          IF    ID    EX   MEM    WB
  Inst 2:                IF    ID    EX   MEM    WB
  Inst 3:                      IF    ID    EX   MEM    WB
  Inst 4:                            IF    ID    EX   MEM    WB

  REG      0  REG      1  REG      2  REG      3  REG      4  REG      5  REG      6  REG      7
  0x00000000  0x00000020  0x00000000  0x000010bb  0x00000c64  0x00000000  0x00000000  0x00000000

  REG      8  REG      9  REG     10  REG     11  REG     12  REG     13  REG     14  REG     15
  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000


  MEM0    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

  MEM8    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

  MEM16   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

  MEM24   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

  MEM32   0x000010bb  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

  MEM40   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

  MEM48   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

  MEM56   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000


  Cycle_index: 9
  Redirection: ON
  RAW_stall: 0
  Control_stall: 0
  Struction_stall: 0
  Branch_inst: 0
  Branch_success_rate: -nan
○ xjtunly@ww:~/system/lab2/pipline5$ 
```

执行结果与实验一的结果完全一致，缺陷是没有显示出定向的数据通路。

实验任务四：控制相关

继续使用上一次饰演的代码，修改main.cpp，且执行如下

```cpp
#include "pip.hpp"
using namespace std;
void model4();
int main(){
    // addu rt, rs, imm          read rs,      write rt
    // add  rd, rs, rt           read rs, rt   write rd
    // lw   rt, rs, imm         read rs,      write rt
    // sw   rt, rs, imm         read rs, rt
    // beqz rs, imm             read rs
    model4();
    return 0;
}
void model4(){
    // ADDU  R1, R0, 32
    // LW    R3, 0(R1)
    // LW    R4, 0(R1)
    // BEQZ  R5, 1
    // ADDU  R3, R3, 4444
    // ADDU  R3, R3, 1111
    // SW    R3, 0(R1)
    vector<Inst> inst;
    inst.push_back(Inst("ADDU", 0, 1, 0, 32));
    inst.push_back(Inst("LW"  , 1, 3, 0, 0));
    inst.push_back(Inst("LW"  , 1, 4, 0, 0));
    inst.push_back(Inst("BEQZ", 5, 0, 0, 1));
    inst.push_back(Inst("ADDU", 3, 3, 0, 4444));
    inst.push_back(Inst("ADDU", 3, 3, 0, 1111));
    inst.push_back(Inst("SW"  , 1, 3, 0, 0));
    pipline p;
    p.storages_init();
    // p.set_redirect();
    for(int i = 0; i < inst.size(); i++){
        p.insts_init(inst[i]);
    }
    p.memory_set(32, 3172);
    p.run_to_the_end();
    p.print_log();
    puts("");
    p.print_reg(16);
    puts("");
    p.print_mem(64);
    puts("");
    p.print_performance();
}
```

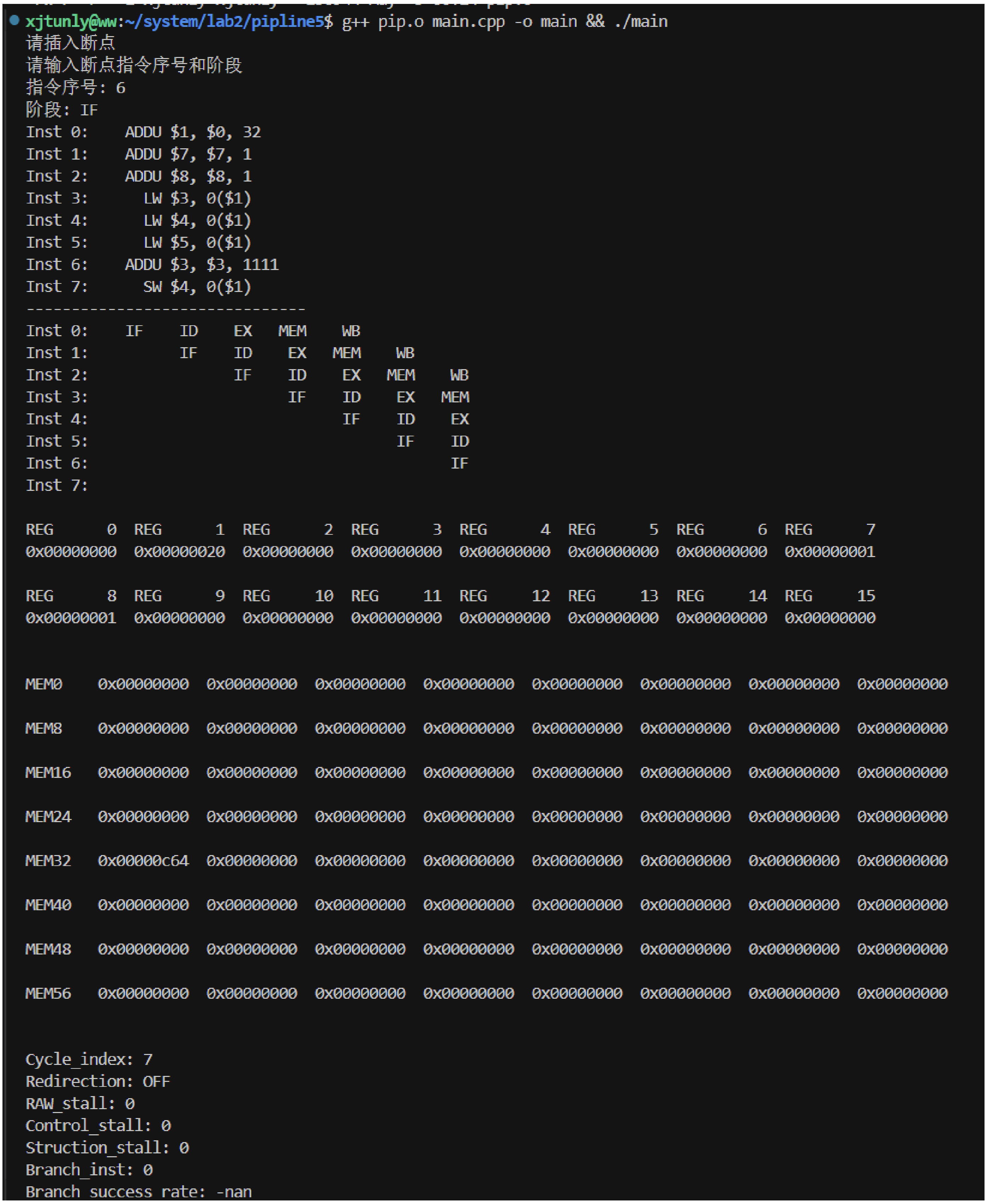


```
xjtunly@ww:~/system/lab2/pipline5$ g++ pip.o main.cpp -o main && ./main
Inst 0:    ADDU $1, $0, 32
Inst 1:      LW $3, 0($1)
Inst 2:      LW $4, 0($1)
Inst 3:    BEQZ $5, 1
Inst 4:    ADDU $3, $3, 4444
Inst 5:    ADDU $3, $3, 1111
Inst 6:      SW $3, 0($1)
-------------------------------
Inst 0:    IF    ID    EX   MEM    WB
Inst 1:          IF   ID* STALL    ID    EX   MEM    WB
Inst 2:                IF STALL STALL    ID    EX   MEM    WB
Inst 3:                                  IF    ID    EX   MEM    WB
Inst 4:
Inst 5:                                              IF    ID    EX   MEM    WB
Inst 6:                                                    IF   ID* STALL    ID    EX   MEM    WB

REG      0  REG      1  REG      2  REG      3  REG      4  REG      5  REG      6  REG      7
0x00000000  0x00000020  0x00000000  0x000010bb  0x00000c64  0x00000000  0x00000000  0x00000000

REG      8  REG      9  REG     10  REG     11  REG     12  REG     13  REG     14  REG     15
0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000


MEM0    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM8    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM16   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM24   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM32   0x000010bb  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM40   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM48   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM56   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000


Cycle_index: 15
Redirection: OFF
RAW_stall: 4
Control_stall: 1
Struction_stall: 1
Branch_inst: 1
Branch_success_rate: 1.000000
```

与上次实验的结果完全一致。

断点执行展示:

> 采用任务一中的代码，要求执行到Inst6的IF段停止，函数编写只需要将任务一的model1()的$p.run_to_the_end()$改为$p.run_to_breakpoint()$，效果如下图:



```
xjtunly@ww:~/system/lab2/pipline5$ g++ pip.o main.cpp -o main && ./main
请插入断点
请输入断点指令序号和阶段
指令序号: 6
阶段: IF
Inst 0:    ADDU $1, $0, 32
Inst 1:    ADDU $7, $7, 1
Inst 2:    ADDU $8, $8, 1
Inst 3:      LW $3, 0($1)
Inst 4:      LW $4, 0($1)
Inst 5:      LW $5, 0($1)
Inst 6:    ADDU $3, $3, 1111
Inst 7:      SW $4, 0($1)
------------------------------
Inst 0:    IF    ID    EX   MEM    WB
Inst 1:          IF    ID    EX   MEM    WB
Inst 2:                IF    ID    EX   MEM    WB
Inst 3:                      IF    ID    EX   MEM
Inst 4:                            IF    ID    EX
Inst 5:                                  IF    ID
Inst 6:                                        IF
Inst 7:

REG      0  REG      1  REG      2  REG      3  REG      4  REG      5  REG      6  REG      7
0x00000000  0x00000020  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000001

REG      8  REG      9  REG     10  REG     11  REG     12  REG     13  REG     14  REG     15
0x00000001  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000


MEM0    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM8    0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM16   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM24   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM32   0x00000c64  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM40   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM48   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000

MEM56   0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000  0x00000000


Cycle_index: 7
Redirection: OFF
RAW_stall: 0
Control_stall: 0
Struction_stall: 0
Branch_inst: 0
Branch success rate: -nan
```

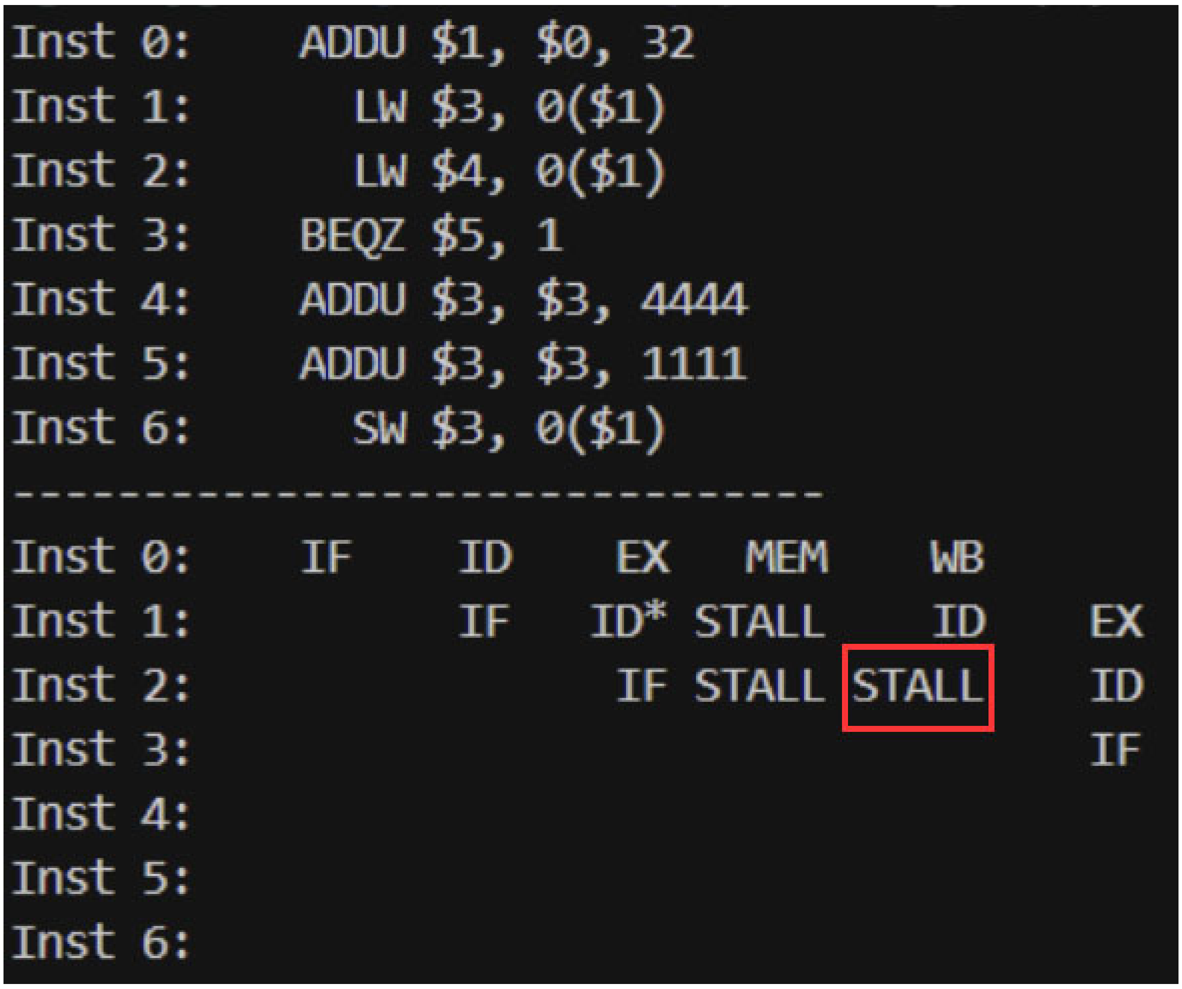
设计思想、特色:

设计模拟器之前，我观察了课本自带的模拟器的执行逻辑与行为，结合所学的知识，将指令的执行阶段分为9个状态，如上文中的表格所示。我的模拟器以时钟周期图为核心，将整个时钟周期图抽象为由若干列记录组成的记录矩阵，每次单步执行都会产生一列新的记录，而要获得一列新的记录，就要读取当前指令的执行状态，并且做出判断，将新的执行状态加入到新的记录中，在一列记录生成结束后，将记录加入到总的时钟周期图末尾，就完成了一次单步执行。

在设计、实现时，我使用指令类中的stage属性存储指令的最新执行阶段，当指令的最新阶段是IF，则尝试指令译码，若在译码过程中发现了RAW冲突，则进入ID_bad状态，这需要在时钟周期图上表示出来，相信读者已经发现，上面的例子演示中，ID_bad被表示为ID*，这表示译码失败，至少需要等待一个时钟周期，否则正常进入ID段；当指令状态是ID时，则表示指令可以执行，进入EX段；若指令处于ID_bad状态，则再次尝试译码，若再次失败，则等待等待一个周期，即加上STALL记录，否则进入ID段；当指令要进入MEM段，需要访存，这一段里，SW指令会将数据放入内存中，其余的指令没有特殊动作；当指令处于MEM完成的状态，则需要写回寄存器，这时ADD、ADDU、LW指令会写自己想要写入的寄存器。

需要注意的是，STALL并不是一个指令状态，因为这不是一个执行的最新状态，相反，ID_bad是一个状态，而且译码失败与未译码不同，译码失败状态下再次译码失败会加上一个STALL记录，而IF之后的译码失败不会立即进入STALL，这符合事实。

此外，我的模拟器有以下的特色部分：在下图中红圈的部分的STALL，应该是由于Inst1在这周期内使用了ID段而导致Inst2无法继续执行，我认为这属于结构冲突，所以在性能统计的代码中，我将这种情况加入到了结构冲突中。



```
Inst 0:    ADDU $1, $0, 32
Inst 1:      LW $3, 0($1)
Inst 2:      LW $4, 0($1)
Inst 3:    BEQZ $5, 1
Inst 4:    ADDU $3, $3, 4444
Inst 5:    ADDU $3, $3, 1111
Inst 6:      SW $3, 0($1)
------------------------------
Inst 0:    IF    ID    EX   MEM    WB
Inst 1:          IF   ID* STALL    ID    EX
Inst 2:                IF STALL STALL    ID
Inst 3:                                  IF
Inst 4:
Inst 5:
Inst 6:
```

在控制冲突的处理中，当发生了BEQZ指令的译码之后，就能决定要继续执行的指令的编号，所以暂停下一条新指令的fetch，即BEQZ指令的ID段执行过程中不能再fetch新的指令。

在处理完上一个log列之后，检查是否能fetch新指令，只要没有STALL发生、没有BEQZ的译码，而且还有新指令等待fetch，就需要将新指令加入到记录中，新指令的编号正好由PC寄存器指明。

其余的格式化输出部分基本没有特色设计。

感悟、缺点

本次实验之初，我对五段流水线的设计存在心里畏惧，其中的状态迁移、STALL的插入都看似极其难以理解，而在手动单步执行了几次代码之后，我能够理解流水线的迁移逻辑，并设计出自己的模拟器。很自豪的一点是，这个模拟器不包含ChatGPT等大语言模型的生成结果。

不足之处也很多，首先支持的指令数量太少，只有简单的LOAD、STORE、ADD、ADDU、BEQZ，但是若要添加其余指令，基本都属于这四种指令类型（LW、SW、R型、跳转）；其次是不支持流水线的倒退，由于倒退可能需要撤销对寄存器、存储器的写入，应该可以采用延迟写入的思想来解决，或者保存以往所有历史状态来暴力解决；最后是不支持指令的文件读取，需要手动编写函数并在main中调用。