Tomasulo 算法仿真器

项目背景

本项目是计算机系统结构课程选题之一，目的是通过手写仿真器来实现Tomasulo算法，加深对Tomasulo算法的理解。

项目原理

Tomasulo 算法中指令执行步骤分为三个阶段：

一、流出

从指令队列的头部取一条指令。若该指令的操作所要求的保留站有空闲（避免结构冲突），就把该指令送到该保留站(设为 r)。若其操作数在寄存器中已经就绪，就将这些操作数送入保留站 r；若其操作数未就绪，就把将产生该操作数的保留站的标识送入保留站 r，一旦被记录的保留站完成计算，直接把数据送给保留站 r。通过寄存器换名和对操作数进行缓冲，消除 WAR 冲突。另外，还要完成对目的寄存器的预约工作。

由于指令是按序流出的，如果有多条指令写同一个结果寄存器时，最后留下的预约结果一定是最后一条指令的，即消除了 WAW 冲突。

二、执行

当两个操作数都就绪后，本保留站就用相应的功能部件开始执行指令规定的操作，把发生 RAW 冲突的可能性减小到最少。load 和 store 指令的执行需要两个步骤：计算有效地址并把有效地址放入 load 或 store 缓冲器，load 指令还要从存储器中读取数据。

三、写结果

功能部件计算完毕后，如果没有其他指令正在写结果，CDB 是就绪的，就将计算结果放到 CDB 上，所有等待该计算结果的寄存器和保留站都同时从 CDB 上获得所需要的数据。store 指令完成对存储器的写入。

技术路线

* 项目框架

1. 变量的定义与初始化

init 函数指明数字与指代的指令的操作码的映射关系和数字与指代的保留站之间的映射关系。

```
int t;

std::queue<int> q;//指令队列（流出队列）
struct st
{
    int n,step;
};
struct instr  //存放指令的各个细节
{
    std::string op;
    int fi,fj,fk,time,device,vj,vk,qj,qk,a;
    int issue,exeution,write;
    int begin,result1,result2,result3,result4,result5,result6,flag1,flag2;
    std::string name;
} instruction[100];

int reg[32];//该寄存器将接收对应保留站的结果
int device[8];//存放占用该保留站的指令号
int result[3][32];   //存放寄存器的值，即保留站device产生的结果
std::set<int> step[6];//存放各个阶段的指令队列
std::set<int>::iterator it,ite;//迭代器，用来枚举
std::map<std::string,int> map;//用来做映射，将数字与指代的指令的操作码联系
std::map<int,std::string> mapp;//用来做映射，将数字与指代的保留站联系
```

```
inline void init()
{
    map["LD"] = 1;map["SUBD"] = 3;map["ADDD"] = 3;map["DIVD"] = 6;map["MULTD"] = 6;
    mapp[1] = "load1";mapp[2] = "load2";mapp[3] = "add1";mapp[4] = "add2";mapp[5] = "add3"; mapp[6] = "mult1"; mapp[7] = "mult2";
}
```

2. read 函数用于读取指令的信息，包括指令的操作码 op、目的寄存器号 fi、源操作数所在的寄存器号 fj 和 fk，以及设置指令执行所需的周期数 time,load 指令:2 个时钟周期；加、减法指令:2 个时钟周期；乘法指令:10 个时钟周期；除法指令:20 个时钟周期。

```
inline void read()
{
    //读入要执行的指令数
    scanf("%d",&t);
    //去除多余换行符，防止读入空行
    char ch;
    while(1)
    {
        ch = getchar();
        if (ch == '\n') break;
    }

    for(int i = 1;i <= t;i++)
    {
        std::string s;
        getline(std::cin,s);
        instruction[i].name = s;
        //处理 LD 指令
```

```
if (s[0] == 'L')
{
    instruction[i].op = "LD";
    instruction[i].time = 2;
    int cnt = 0;
    for(int j = 2;j < s.length();j++)
    {
        if (s[j] != ' ')
        {
            cnt = j;
            break;
        }
    }
    instruction[i].fi = s[cnt + 1] -'0';
    if (s[cnt + 2] >= '0' && s[cnt + 2] <= '9')
    {
        instruction[i].fi *= 10;
        instruction[i].fi += s[cnt + 2] - '0';
    }

    for(int j = cnt + 1;j < s.length();j++)
    {
        if (s[j] == ',')
        {
            cnt = j;
            break;
        }
    }
    instruction[i].a = s[cnt + 1] - '0';
    if (s[cnt + 2] >= '0' && s[cnt + 2] <= '9')
    {
        instruction[i].a *= 10;
        instruction[i].a += s[cnt + 2] - '0';  //立即数是两位数的情况
    }

    for(int j = cnt + 1;j < s.length();j++)
    {
        if (s[j] == 'R')
        {
            cnt = j;
            break;
        }
    }
    instruction[i].fj = s[cnt + 1] - '0';
```

```
    if (s[cnt + 2] >= '0' && s[cnt + 2] <= '9')
    {
        instruction[i].fj *= 10;
        instruction[i].fj += s[cnt + 2] - '0';
    }
}
//处理其他指令
else
{
    int cnt = 0;
    if (s[0] == 'M')
    {
        instruction[i].op = "MULTD";
        instruction[i].time = 10;
    }
    else if (s[0] == 'S')
    {
        instruction[i].op = "SUBD";
        instruction[i].time = 2;
    }
    else if (s[0] == 'A')
    {
        instruction[i].op = "ADDD";
        instruction[i].time = 2;
    }
    else if (s[0] == 'D')
    {
        instruction[i].op = "DIVD";
        instruction[i].time = 20;
    }
    for(int j = 1;j < s.length();j++)
    {
        if (s[j] == 'F')
        {
            cnt = j;
            break;
        }
    }
    instruction[i].fi = s[cnt + 1] -'0';
    if (s[cnt + 2] >= '0' && s[cnt + 2] <= '9')
    {
        instruction[i].fi *= 10;
        instruction[i].fi += s[cnt + 2] - '0';
    }
```

```
            for(int j = cnt + 1;j < s.length();j++)
            {
                if (s[j] == 'F')
                {
                    cnt = j;
                    break;
                }
            }
            instruction[i].fj = s[cnt + 1] -'0';
            if (s[cnt + 2] >= '0' && s[cnt + 2] <= '9')
            {
                instruction[i].fj *= 10;
                instruction[i].fj += s[cnt + 2] - '0';
            }
            for(int j = cnt + 1;j < s.length();j++)
            {
                if (s[j] == 'F')
                {
                    cnt = j;
                    break;
                }
            }
            instruction[i].fk = s[cnt + 1] -'0';
            if (s[cnt + 2] >= '0' && s[cnt + 2] <= '9')
            {
                instruction[i].fk *= 10;
                instruction[i].fk += s[cnt + 2] - '0';
            }
        }
    }
}
```

3. print 函数用于输出每个周期的指令状态表、保留站内容和寄存器状态表的信息。

```
inline void print(int x)
{
    printf("                                                周期%d:\n",x);
    //输出指令状态表
    printf("指令状态:\n");
    printf("指令名        |Issue|Execution completed|Write Result|\n");
    for(int i = 1;i <= t;i++)
    {
        std::cout << instruction[i].name;
        for(int j = 1;j <= 14 - instruction[i].name.length();j++) printf(" ");
        printf("|");
```

```
    if (instruction[i].issue != 0) printf("%5d|",instruction[i].issue);
    else
    {
        for(int j = 1;j <= 5;j++) printf(" ");
        printf("|");
    }

    if (instruction[i].exeution != 0) printf("%19d|",instruction[i].exeution); //执行完毕
    else if(instruction[i].begin>0){
        for(int j = 1;j <= 11;j++) printf(" ");
        printf("正在执行");
        printf("|");
    }
    else
    {
        for(int j = 1;j <= 19;j++) printf(" "); //还未开始执行
        printf("|");
    }

    if (instruction[i].write != 0) printf("%12d|",instruction[i].write);
    else
    {
        for(int j = 1;j <= 12;j++) printf(" ");
        printf("|");
    }
    printf("\n");
}
printf("\n");
//输出保留站内容
printf("保留站内容:\n");
printf("保留站名称|Busy |Op   |Vj           |Vk           |Qj   |Qk   |A         |\n");
for(int i = 1;i <= 7;i++)
{
    std::cout << mapp[i];
    for(int j = 1;j <= 10 - mapp[i].length();j++) printf(" ");
    printf("|");
    if(device[i] != 0 && instruction[device[i]].write == x)
    {
        printf("no   |     |             |             |     |     |          |\n");
        continue;
    }
    else if (device[i] != 0 && instruction[device[i]].write != x) printf("yes  |");
    else
    {
```

```
            printf("no   |     |             |             |     |     |          |\n");
            continue;
        }
        std::cout << instruction[device[i]].op;
        for(int j = 1;j <= 5 - instruction[device[i]].op.length();j++) printf(" ");
        printf("|");
        if (instruction[device[i]].op[0] == 'L')
        {
            //检测第一操作数是否就绪
            if(reg[instruction[device[i]].fj] == 0)
            printf("R[R%d]        |             |0    |     ",instruction[device[i]].fj);
            else{
            printf("             |             |");
            std::cout << mapp[reg[instruction[device[i]].fj]];   //要写这个 fj 寄存器的 device 保留站编号  用 mapp 从编号映射出保留站名称字符串
            for(int j = 1;j <= 5 - mapp[reg[instruction[device[i]].fj]].length();j++) printf(" ");
            printf("|     ");
            }

            //printf("%d,%d",instruction[device[i]].begin,instruction[device[i]].exeution);
            if    (instruction[i].begin    >    0    ||    instruction[i].exeution    !=    0) printf("|R[R%d]+%-4d|",instruction[device[i]].fj,instruction[device[i]].a); //正在执行 计算有效地址放入 A 字段
            else printf("|%-10d|",instruction[device[i]].a);     //最后放入 a 保证字段对齐
        }

        else
        {   //分别判断第一第二操作数

    //printf("%d,%d,%d",instruction[device[i]].result1,instruction[device[i]].result2,instruction[device[i]].result3);
            if(instruction[device[i]].flag1 == 1 && instruction[device[i]].qj == 0){     //vj 字段          //接收保留站产生的结果
                if(instruction[device[i]].result3==1) printf("Mem[R[R%d]+%d]|",instruction[device[i]].result1,instruction[device[i]].result2);
                else       if(instruction[device[i]].result3==2)       printf("R[F%d]+R[F%d]|",instruction[device[i]].result1,instruction[device[i]].result2);
                else       if(instruction[device[i]].result3==3)       printf("R[F%d]-R[F%d]|",instruction[device[i]].result1,instruction[device[i]].result2);
                else       if(instruction[device[i]].result3==4)       printf("R[F%d]*R[F%d]|",instruction[device[i]].result1,instruction[device[i]].result2);
                else       if(instruction[device[i]].result3==5)       printf("R[F%d]/R[F%d]|",instruction[device[i]].result1,instruction[device[i]].result2);
```

```
            }

            else if(instruction[device[i]].flag1 != 1 && instruction[device[i]].qj == 0){

    if(result[2][instruction[device[i]].fj]!=0){
           //从寄存器中取刚被写入的数据
                if(result[2][instruction[device[i]].fj]==1)
printf("Mem[R[R%d]+%d]|",result[0][instruction[device[i]].fj],result[1][instruction[device[i]
].fj]);
                else    if(result[2][instruction[device[i]].fj]==2)    printf("R[F%d]+R[F%d]
|",result[0][instruction[device[i]].fj],result[1][instruction[device[i]].fj]);
                else    if(result[2][instruction[device[i]].fj]==3)    printf("R[F%d]-R[F%d]
|",result[0][instruction[device[i]].fj],result[1][instruction[device[i]].fj]);
                else    if(result[2][instruction[device[i]].fj]==4)    printf("R[F%d]*R[F%d]
|",result[0][instruction[device[i]].fj],result[1][instruction[device[i]].fj]);
                else    if(result[2][instruction[device[i]].fj]==5)    printf("R[F%d]/R[F%d]
|",result[0][instruction[device[i]].fj],result[1][instruction[device[i]].fj]);
                }
                else   printf("R[F%d]                          |",instruction[device[i]].fj);
//从寄存器中取已就绪的数据
            }
            else
            printf("             |");


    //printf("%d,%d,%d,%d,%d,%d",instruction[device[i]].flag2,device[i],instruction[device[i]
].qk,instruction[device[i]].device,instruction[device[i]].fk,result[2][instruction[device[i]]
.fk]);

            if(instruction[device[i]].flag2 == 1 && instruction[device[i]].qk == 0){      //vk
字段
                if(instruction[device[i]].result6==1)
printf("Mem[R[R%d]+%d]|",instruction[device[i]].result4,instruction[device[i]].result5);
                else       if(instruction[device[i]].result6==2)       printf("R[F%d]+R[F%d]
|",instruction[device[i]].result4,instruction[device[i]].result5);
                else       if(instruction[device[i]].result6==3)       printf("R[F%d]-R[F%d]
|",instruction[device[i]].result4,instruction[device[i]].result5);
                else       if(instruction[device[i]].result6==4)       printf("R[F%d]*R[F%d]
|",instruction[device[i]].result4,instruction[device[i]].result5);
                else       if(instruction[device[i]].result6==5)       printf("R[F%d]/R[F%d]
|",instruction[device[i]].result4,instruction[device[i]].result5);
            }

            else if(instruction[device[i]].flag2 != 1 && instruction[device[i]].qk == 0){
```

```
                if(result[2][instruction[device[i]].fk]!=0){
                if(result[2][instruction[device[i]].fk]==1)
printf("Mem[R[R%d]+%d]|",result[0][instruction[device[i]].fk],result[1][instruction[device[i]
].fk]);
                else    if(result[2][instruction[device[i]].fk]==2)    printf("R[F%d]+R[F%d]
|",result[0][instruction[device[i]].fk],result[1][instruction[device[i]].fk]);
                else    if(result[2][instruction[device[i]].fk]==3)    printf("R[F%d]-R[F%d]
|",result[0][instruction[device[i]].fk],result[1][instruction[device[i]].fk]);
                else    if(result[2][instruction[device[i]].fk]==4)    printf("R[F%d]*R[F%d]
|",result[0][instruction[device[i]].fk],result[1][instruction[device[i]].fk]);
                else    if(result[2][instruction[device[i]].fk]==5)    printf("R[F%d]/R[F%d]
|",result[0][instruction[device[i]].fk],result[1][instruction[device[i]].fk]);
                }
                else printf("R[F%d]        |",instruction[device[i]].fk);
            }

            else
            printf("             |");

            if(instruction[device[i]].qj == 0)  //qj 字段
            printf("0    |");
            else{
            //printf("%d",instruction[device[i]].qj);
            std::cout << mapp[instruction[device[i]].qj];
            for(int j = 1;j <= 5 - mapp[instruction[device[i]].qj].length();j++) printf(" ");
            printf("|");
            }

            if(instruction[device[i]].qk == 0)  //qk 字段
            printf("0    |");
            else{
            //printf("%d",instruction[device[i]].qk);
            std::cout << mapp[instruction[device[i]].qk];
            for(int j = 1;j <= 5 - mapp[instruction[device[i]].qk].length();j++) printf(" ");
            printf("|");
            }
            printf("          |");     //最后保证 a 字段对齐
        }
        printf("\n");
    }
    printf("\n");
    //输出寄存器状态表
    printf("寄存器状态表:\n");
    printf("寄存器号|");
```

```
    for(int i = 0;i <= 12;i+=2)
        {
            printf("F%-12d|",i);
        }
    printf("...|F30  |\n");

    printf("Qi      |");
    for(int i = 0;i <= 12;i+=2)
        {
            if (reg[i] != 0)
                    {
                    std::cout << mapp[reg[i]];
                    for(int j = 1;j <= 13 - mapp[reg[i]].length();j++) printf(" ");
                    }
            else printf("            ");
            printf("|");
        }
    printf("...|F30  |\n");

    printf("值      |");
    for(int i = 0;i <= 12;i+=2)
        {
            if(reg[i]            ==            0            &&            result[2][i]==1)
printf("Mem[R[R%d]+%d]",result[0][i],result[1][i]);
            else    if(reg[i]    ==    0    &&    result[2][i]==2)    printf("R[F%d]+R[F%d]
",result[0][i],result[1][i]);
            else    if(reg[i]    ==    0    &&    result[2][i]==3)    printf("R[F%d]-R[F%d]
",result[0][i],result[1][i]);
            else    if(reg[i]    ==    0    &&    result[2][i]==4)    printf("R[F%d]*R[F%d]
",result[0][i],result[1][i]);
            else    if(reg[i]    ==    0    &&    result[2][i]==5)    printf("R[F%d]/R[F%d]
",result[0][i],result[1][i]);

//if(reg[i] ! = 0 && result[2][i]==1) printf("Mem[R[R%d]+%d]",result[0][i],result[1][i]);


            else printf("            ");
            printf("|");
        }
    printf("...|F30  |\n");
}
```

4. procedure 函数用于实现算法具体流程，在每个周期按顺序判断每条指令的状态。

如果有空闲保留站，指令按序流出到保留站，根据寄存器状态表的值是否为 0 判断源操作数是否就绪，进行寄存器换名，修改保留站对应字段，预约目的寄存器。

如果指令的操作数都就绪就进入执行阶段，进行运算，否则指令停顿。

当执行达到规定的执行周期数，并且在这个周期没有其他指令正在写结果，CDB 是就绪状态时，指令进入写结果阶段，遍历保留站的 Qj、Qk 表以及寄存器状态表，将运算结果传送到需要该结果的地方，并释放保留站资源。

```
inline void procedure()
{
    for(int i = 1;i <= t;i++) q.push(i); //所有指令按顺序进入队列
    int time = 0;
    while(1)
    {
        system("Pause");

    printf("----------------------------------------------------------------------------------------
----------------------------------------\n");
        std::queue<st> now;//储存当前周期的队列变换 st 是结构体 储存 n 和 step
        time++;
        //流出阶段
        if (!q.empty())
        {
            int x = q.front(); //从指令队列头部取一条指令
            //寻找是否存在空闲保留站
            if (map[instruction[x].op] == 1)
            {
                if (device[1] == 0) instruction[x].device = 1;
                else if (device[2] == 0) instruction[x].device = 2;
            }
            else if (map[instruction[x].op] == 3)
            {
                if (device[3] == 0) instruction[x].device = 3;
                else if (device[4] == 0) instruction[x].device = 4;
                else if (device[5] == 0) instruction[x].device = 5;
            }
            else if (map[instruction[x].op] == 6)
            {
                if (device[6] == 0) instruction[x].device = 6;
                else if (device[7] == 0) instruction[x].device = 7;
            }
            //分配保留站 把代表该指令的编号放到指令 x 的结构体的 device 中

            if (instruction[x].device != 0)
            {

                    instruction[x].issue = time; //可以流出
                    reg[instruction[x].fi] = instruction[x].device; //即将要写目的寄存器的指
```

令是 x reg 里面存的是保留站号

```
                    if (reg[instruction[x].fj] == 0) instruction[x].qj = 0; //第一操作数就绪
                    else
                    {
                        instruction[x].qj = reg[instruction[x].fj];
                    }

                    if (reg[instruction[x].fk] == 0) instruction[x].qk = 0; //第二操作数就绪  如果没有指令要写 fk 寄存器 qk 为 0
                    else
                    {
                        instruction[x].qk = reg[instruction[x].fk]; //要写 fk 寄存器的指令的保留站名称放到 qk
                    }
                    device[instruction[x].device] = x;
                    //将流出阶段完成的 x 放入执行队列 包括待执行以及正在执行的指令
                    now.push((st){x,2});
                    q.pop();
            }
        }
        //执行阶段
        for(it = step[2].begin();it != step[2].end();++it)
        {
            int x = *it;
            //判断两个源操作数是否就绪
            if (instruction[x].op[0] == 'L' ||(instruction[x].qj==0 && instruction[x].qk==0))
            {
                instruction[x].begin++;
                instruction[x].time--;

                if (instruction[x].time == 0)
            {
                instruction[x].exeution = time; //达到所需的执行周期，执行完毕之后，将执行完的时钟周期数放到指令的结构体的 exeution 里保存
                instruction[x].begin=0;
                //将执行阶段完成的指令 x 放入写结果队列，并从执行队列删除
                now.push((st){x,3});
            }
            }
        }
         //写结果阶段
        for(it = step[3].begin();it != step[3].end();++it)
        {
            int x = *it;  //printf("%d",x); //x 为指令号，代表正在写结果的指令
```

```
                instruction[x].write = time;
                for(int i = 0;i <= 12;i+=2){
                    if(reg[i] == instruction[x].device){
                    result[0][i]=instruction[x].fj;
                    if(instruction[x].op=="LD") result[1][i]=instruction[x].a;
                    else result[1][i]=instruction[x].fk;
                    if (instruction[x].op == "LD") result[2][i]=1;
                    else if (instruction[x].op == "ADDD") result[2][i]=2;
                    else if (instruction[x].op == "SUBD") result[2][i]=3;
                    else if (instruction[x].op == "MULTD") result[2][i]=4;
                    else if (instruction[x].op == "DIVD") result[2][i]=5;
                }
                }
                for(int i = 0;i <= 12;i+=2){
                    if(reg[i] == instruction[x].device) reg[i] = 0;
                }
                device[instruction[x].device] = 0;

                for (int i = 1; i <= 7; i++) {
                    if(device[i]!=0){
                    if (instruction[device[i]].qj == instruction[x].device) {
                            instruction[device[i]].result1 = instruction[x].fj;
                            if(instruction[x].op=="LD")
instruction[device[i]].result2=instruction[x].a;
                    else instruction[device[i]].result2 = instruction[x].fk;

                    if (instruction[x].op == "LD") {instruction[device[i]].result3=1; }
                    else if (instruction[x].op == "ADDD") {instruction[device[i]].result3=2;}
                    else if (instruction[x].op == "SUBD") {instruction[device[i]].result3=3;}
                    else if (instruction[x].op == "MULTD") {instruction[device[i]].result3=4;}
                    else if (instruction[x].op == "DIVD") {instruction[device[i]].result3=5;}
                    instruction[device[i]].flag1 = 1;
                    instruction[device[i]].qj = 0;
                        }
                        if (instruction[device[i]].qk == instruction[x].device) {
                            instruction[device[i]].result4 = instruction[x].fj;
                            if(instruction[x].op=="LD")
instruction[device[i]].result5=instruction[x].a;
                    else instruction[device[i]].result5 = instruction[x].fk;

                    if (instruction[x].op == "LD") {instruction[device[i]].result6=1;}
                    else if (instruction[x].op == "ADDD") {instruction[device[i]].result6=2;}
                    else if (instruction[x].op == "SUBD") {instruction[device[i]].result6=3;}
                    else if (instruction[x].op == "MULTD") {instruction[device[i]].result6=4;}
```

```
                else if (instruction[x].op == "DIVD") {instruction[device[i]].result6=5;}

                instruction[device[i]].qk = 0;
                instruction[device[i]].flag2 = 1;
                    }
            }
            }
            //将写结果阶段完成的 x 放入结束队列，并从写结果队列中删除
            now.push((st){x,4});
            break;  //控制一个周期最多只有一条指令写结果
        }

            //执行该指令周期的队列更新
        while(!now.empty())
        {
            st x = now.front();
            now.pop();
            step[x.step - 1].erase(x.n);
            step[x.step].insert(x.n);
        }
        //输出结果
        print(time);
        //判断是否所有指令均已完成
        if (step[4].size() == t) break;
    }
}
```

* 关键技术

* 状态机

每条指令有三个状态：流出、执行和写结果。当指令满足进入条件时，进入下一状态。

如果有空闲保留站，指令进入流出状态，按序流出到保留站，根据寄存器状态表的值是否为 0 判断源操作数是否就绪，修改保留站对应字段，预约目的寄存器。如果指令的操作数都就绪就进入执行状态，进行运算。当执行达到规定的执行周期数，并且在这个周期没有其他指令正在写结果，CDB 是就绪状态时，指令进入写结果状态，遍历保留站的 Qj、Qk 表以及寄存器状态表，将运算结果传送到需要该结果的地方，并释放保留站资源。

关键代码：

```
for(int i = 1;i <= t;i++) q.push(i); //所有指令按顺序进入流出队列
std::queue<st> now;//储存当前周期的队列变换 st 是结构体 储存 n（指令号）和 step（所处第几阶段）
std::set<int> step[6];//存放各个阶段的指令队列
//流出阶段
int x = q.front(); //从指令队列头部取一条指令，寻找是否存在空闲保留站，如果有空闲的保留站，分
```

配保留站并设置指令的 issue 值为当前周期数，指令可以流出
now.push((st){x,2});//将流出阶段完成的 x 放入执行队列，包括待执行以及正在执行的指令
q.pop();
//执行阶段
for(it = step[2].begin();it != step[2].end();++it)
{
    int x = *it; //x 为指令号，代表正在执行队列的的指令
    //判断两个源操作数是否就绪，如果都已就绪，指令状态为正在执行，当达到所需的执行周期，执行完毕之后，设置指令的 exeution 值为当前时钟周期数：instruction[x].exeution = time;
    now.push((st){x,3});//将执行阶段完成的指令 x 放入写结果队列，并从执行队列删除
}
//写结果阶段
for(it = step[3].begin();it != step[3].end();++it)
{
    int x = *it; //x 为指令号，代表正在写结果的指令
    instruction[x].write = time;//设置指令的 write 值为当前时钟周期数
    now.push((st){x,4});//将写结果阶段完成的 x 放入结束队列，并从写结果队列中删除
    break; //控制一个周期最多只有一条指令写结果
}
//执行该指令周期的队列更新
        while(!now.empty())
        {
            st x = now.front();
            now.pop();
            step[x.step - 1].erase(x.n);
            step[x.step].insert(x.n);
        }
//输出结果
        print(time);
//判断是否所有指令均已完成
        if (step[4].size() == t) break;

* 指令循环

通过 while(1)大循环判断每个周期指令的状态，read 函数中使用 for 循环读取每条指令的操作码、操作数等信息，对于每个阶段对应的指令队列中，使用 for 循环和迭代器 it 遍历指令并修改相应的值。

* 整数与字符串之间的映射

std::map<std::string,int> map;//用来做映射，将数字与指代的指令的操作码联系
std::map<int,std::string> mapp;//用来做映射，将数字与指代的保留站联系
map["LD"] = 1;map["SUBD"] = 3;map["ADDD"] = 3;map["DIVD"] = 6;map["MULTD"] = 6;
mapp[1] = "load1";mapp[2] = "load2";mapp[3] = "add1";mapp[4] = "add2";mapp[5] = "add3"; mapp[6]

= "mult1"; mapp[7] = "mult2";
如当 map[instruction[x].op] == 1 时，代表该指令的操作码是 LD，所对应的是 load 缓冲器。
再比如在 print 函数中 std::cout << mapp[reg[instruction[device[i]].fj]]; //用 mapp 做映射，从要写 fj 寄存器的保留站的编号映射出保留站名称（字符串）。

* 写结果时 CDB 就绪态的判断

在写结果阶段在 for 循环中使用 break，只从写结果队列取一条指令，将保留站产生的结果传给寄存器和保留站的对应字段中，然后 break 跳出循环来保证每个周期最多只有一条指令写结果。

项目实现

本项目使用 Dev-C++编写代码，可以在该环境中编译运行。编程语言为 C 语言和 C++。

工作量

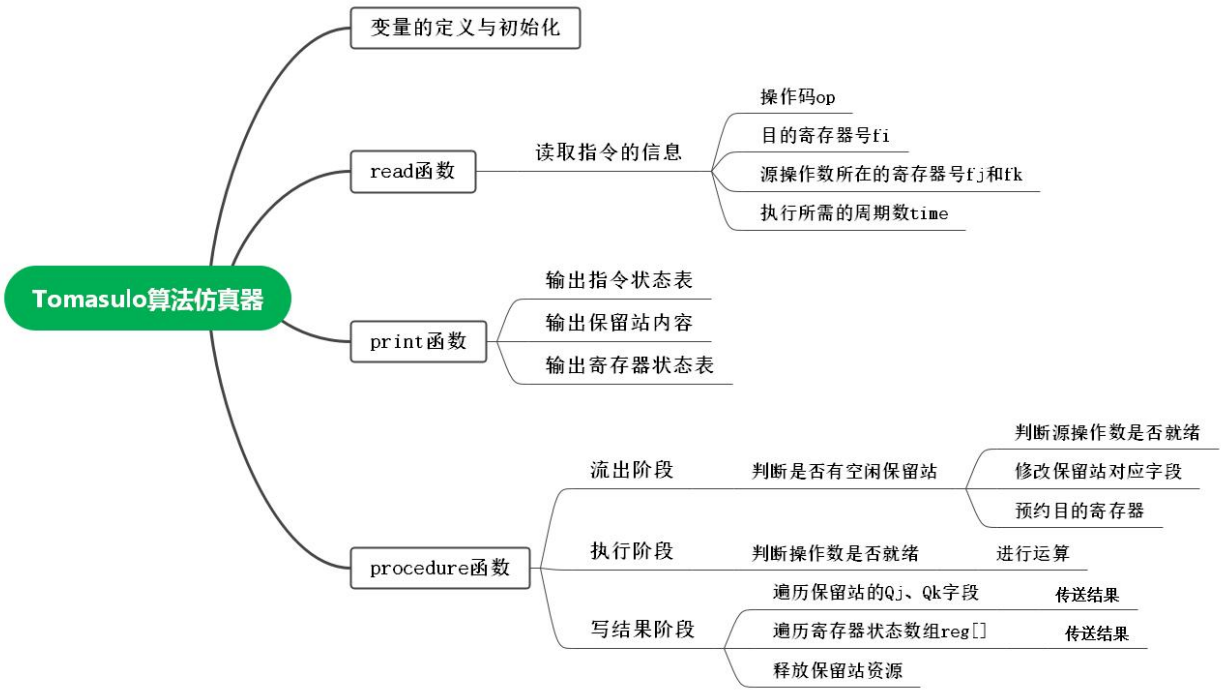
* 代码行数：524 行
* 时间：共计约 34.5 小时

时间记录日志

日期	开始时间	结束时间	净时间	活动
6.2	11:00	12:00	1 小时	配置环境 Dev-C++，解决中文字符不显示的问题
6.3	14:20	17:40	3 小时 20 分钟	理清算法大致思路与框架
6.5	17:24	23:30	6 小时 6 分钟	写读指令和输出结果两个函数
6.8	14:30	18:00	3 小时 30 分钟	编写流出阶段代码，将寄存器号换名为保留站编号或者操作数
6.10	17:00	23:00	6 小时	使用 result 二维数组保存保留站的结果，在写结果阶段传入等待该结果的的寄存器中存储
6.11	8:30	10:00	1 小时 30 分钟	写结果阶段使用 instruction[x].result1 等六个变量存储保留站的结果并传入保留站 Qj、Qk 字段需要的位置；与老师沟通交流
6.12	20:30	22:00	1 小时 30 分钟	在写结果阶段使用 break 来保证每个周期最多只有一条指令写结果
6.13	11:00	15:00	4 小时	解决写结果阶段传值的问

				题；进行测试；修复 bug
6.13	19:00	23:00	4 小时	完成技术文档项目背景、原理、框架部分
6.14	16:30	19:30	3 小时	完成技术文档中关键技术和工作量部分
6.15	9:00	10:00	1 小时	上传项目至 github

WBS 工作任务分解图



测试用例

本项目给出了典型的测试用例，例如具有写后写冲突、读后写冲突和写后读冲突的指令，可以体现 Tomasulo 算法的核心思想，即把发生写后读冲突的可能性减小到最少，以及通过保留站实现寄存器换名，消除写后写冲突和读后写冲突。

```
//用例 1：写后写冲突
2
ADDD F2,F6,F4
LD F2,45(R3)

//用例 2：读后写冲突
3
MULTD F0,F2,F4
MULTD F6,F8,F0
ADDD F0,F8,F2
```

```
//用例 3：写后读冲突
3
LD F6,34(R2)
LD F6,45(R3)
ADDD F0,F2,F6
```

参考资料

《计算机系统结构教程（第二版）》张晨曦、王志英等编著