实验二：Tomasolu

计71 韩荣 2016010189

1、 实验目的

1、理解流水线中乱序执行的逻辑

2、了解并实现Tomasolu算法

3、增加Tomasolu算法的功能，如添加JUMP指令。

4、与计分牌算法进行比较

2、 实验要求

实验的基本要求为设计实现不带JUMP的Tomasolu算法，输出其log，并对其进行性能测试。

实验的拓展要求为设计实现JUMP的Tomasolu算法，输出其log。

3、运行方法与设计思路

实验环境

MACOS 10.15.1，Apple clang version 11.0.0 (clang-1100.0.33.8)

编译方法

可以使用Makefile，直接make后运行main即可，需要注意命令行参数的使用：

运行方法

```
./main basic 0/1/2/3/4
```

上述命令第一个参数输入“basic”，第二个参数输入序号，可以对basic进行测试和log输出，Cycle信息打印在屏幕上。

```
./main extend filename
```

上述命令第一个参数输入“extend”，第二个参数输入序号，可以对其他文件进行测试和log输出，Cycle信息打印在屏幕上。

实验思路

本次实验一共使用了如下类：

Instruction.h

```
class Instruction{  //指令类，用于记录指令信息
public:
    std::string Op;
    std::string Dst;
    std::string Src1;
    std::string Src2;
    // 记录指令运行时间
    int Issue;
    int ExecComp;
    int WriteResult;
    int Count;
    int fill;
    explicit Instruction(std::string instr);
};
```

Simulator.h

```
class rs{       //保留站的一项
public:
    explicit rs();
    std::string Busy;
    std::string Op;
    std::string Vj;
    std::string Vk;
    std::string Qj;
    std::string Qk;
    int Instr;
    int Allocate;
    int TimeLeft;
    int ExecTime;
    int Status;
    void issue();
    void exec();
    void update();
    void done();
    void writeback();
};
class lb{           //Load Buffer的一项
public:
    explicit lb();
    std::string Busy;
    std::string Address;
    int Instr;
    int Allocate;
    int ExecTime;
```

```
    int TimeLeft;
    int Status;
    void issue();
    void exec();
    void update();
    void done();
    void writeback();
};
class RStation{     //保留站
public:
    rs Ars[6];
    rs Mrs[3];
    void Print();
};
class LoadBuffer{   //Load Buffer
public:
    lb LB[3];
    void Print();
};
class Register{     //Register
public:
    explicit Register();
    std::string R[32];
    void Print();
};
class FU{           //FU
public:
    explicit FU();
    int ld[2];
    int add[3];
    int mul[2];
};
class Simulator{    //Simulator
private:
    std::string IS;
    std::string Ready;
    std::string Done;
    std::string WB;
    RStation RS;
    LoadBuffer L;
    Register R;
    std::vector<Instruction> Instructions;
    FU resource;
public:
    int Cycle;
    int CanIssue;
```

```
    std::string filename;
    std::string stofill;
    int IssuePointer;
    explicit Simulator();
    bool CheckNumber(std::string s);
    bool CheckComplete(rs instr);
    bool CheckLoadComplete(lb instr);
    void Work(std::string file,std::string log);
    void TryIssue(int pointer);
    void Exec();
    void TryExec();
    int IsVacant(std::string op,int pointer);
    bool FindPlace(int pointer);
    bool NotFull();
    void Tomasolu();
    void Print();
    void PrintLog();
    void Clean(std::string op,int fill);
    double WriteBack();
    void ShowInstrStatus();
};
```

其中最重要的是Simulator类，用于模拟器的实现，包括Tomasolu算法的各个部分，以及各个保留站和FU的协调调度。

根据Tomasolu算法，在Simulator类中实现一个Tomasolu函数，可以将一个周期的内容分为如下几个部分：

```
void Simulator::Tomasolu(){
    while(NotFull()){   //如果所有指令没有都至少完成一次
        WriteBack();        //写回
        TryIssue(IssuePointer);   //发射
        Exec(); //执行
        TryExec();  //检查是否就绪
        Print();    //打印Cycle信息
        Cycle += 1; //进入下一个Cycle
    }
}
```

写回:可以发现这个算法首先进行的是写回操作，在写回操作中，首先需要将结果计算出来，并更新保留站中等待这个结果的信息，之后如果寄存器中该寄存器对应的保留站与写回指令相同，则写回寄存器，否则不写，避免WAW冲突。最后，将相关的FU和RS/LD Buffer信息清空，方便下一条指令进行操作。

发射：Simulator类中维护一个IssuePointer，这个IssuePointer指向下一条将要发射的指令。发射时需要检查是否还有剩余的保留站，如果有的话，就直接发射，并将目标寄存器中的值修改为对应保留站的内容。值得注意的是，由于寄存器重命名，所以无论上一个是否已经写回，都可以直接修改目标寄存器中的值。

执行：对所有正处于执行状态的保留站中的剩余时间-1，需要处理两种特殊情况：1、执行完毕，则跳转到完毕状态并记录下这条指令的Exec Comp，2、除法的除数为0，则直接结束除法，跳转到完毕状态并记录下这条指令的Exec Comp。

检查是否就绪：因为写回的原因，可能会改变保留站中指令的就绪情况，对他们进行检查，如果Vj，Vk都具备，则可以进入就绪状态，在下一周期进行执行。

打印Cycle信息：便于打印每个Cycle的详细信息。

以上便是Tomasolu Basic算法的大致思路，通过上述思路可以完成一个不带JUMP指令的Tomasolu算法。

带JUMP的Tomasolu算法设计

为了实现JUMP指令，需要将其放在Ars保留站中，且在Simulator类中额外增加一个信息量CanIssue，在Instruction中也增加一个计数量Count，仅有在Count == 1时，更新Instruction的Issue、Exec Comp、Write Back值，实现只记录第一次运行指令的信息。

JUMP指令的本质和RS中的运算相似，也可以按照Tomasolu算法的上述五个流程进行，需要注意的是，在Issue阶段，它是不会修改Register的值的，并且会关闭CanIssue，阻塞发射，在Write Back阶段，它可能会修改IssuePointer的值，会打开CanIssue，重新允许发射。

4、与计分牌算法的异同

计分牌算法是对实际的FU进行操作，可能会存在WAW的情况导致需要等待很长时间才能发射，而Tomasolu算法可以根据重命名直接进行发射，只要还剩RS即可发射。

两者的大体形式都是相似的。

5、实验结果及分析

basic部分的五个实验结果已经在log中给出，可以发现运行的效率较高，且结果是正确的。

对于performance的部分，经过测试，两个的文件CPI：

MUL.nel = 0.197

Big_test = 0.5

6、实验小结

本次实验对于Tomasolu算法的理解更加透彻了，开始以Instruction为主进行TimeLeft的计算等操作，能够完成Basic，但在性能测试的时候，表现较为糟糕，后来便使用RS为主的计算方式，大大提高了效率。

同时，由于没有参考任何代码，本次实验从构思到实现都经过了较为仔细的思考过程，收获颇丰，对于Tomasolu的掌握也很不错。

感谢助教和老师的付出！