Tomasulo 算法模拟实验报告

计 76 陈之杨 2017011377

2020.5

1 使用方法

`main.cpp` 为源代码，使用 `g++ -o main main.cpp -O2` 命令编译。运行 `main` 进行 Tomasulo 算法模拟。使用标准输入读入汇编指令，标准输出打印 Log 信息。

2 实现细节

代码中实现了三个类。`Operation` 表示指令，存储了指令的内容以及需要输出的指令执行周期信息。`Register` 表示寄存器，记录寄存器中存储的值，或者正在等待执行写回的指令。使用一个 `map<string, Register>` 维护所有寄存器。`ReservationStation` 表示保留站，存储等待执行和执行中的指令信息。特别地，Load Buffer 也视为一种保留站。实现了 `set_reg` 函数使保留站获取当前需要的运算数，`compute` 函数计算指令的结果，`update_value` 函数在一条指令计算出结果后更新其它保留站。`compute_cycle` 函数计算一条指令需要的周期数，另外还设置了 `print_status` 接口打印任意周期的算法状态。`tomasulo` 函数为算法的主函数，按照执行指令（并写回）—发射指令—检查指令就绪的顺序执行。

算法实现流程如下：

(1) 检查所有的保留站，如果存在当前周期结束执行的指令，则标记指令完成周期。如果存在当前周期写回的指令，则修改状态，更新其它保留站，判断是否写入寄存器。

(2) 依次检查所有保留站，如果还有空闲，发射当前的指令。

(3) 如果所有指令都已执行完毕，则结束算法。

(4) 依次检查所有保留站，如果计算资源还有空闲，找到就绪时间为第一关键字，指令编号为第二关键字最小的就绪指令，开始执行。

(5) 周期数加一，结束当前周期，转 1。

3 算法讨论

Tomasulo 作为动态调度算法，通过设置 Reservation Station 保存发射的指令，按照指令就绪的顺序，而非按照代码的正常顺序执行，并将 RS 视作一种寄存器重命名方法解决读后写、写后写冲突。指令的冲突检测和执行控制被分开，指令的执行单纯由 RS 控制。

相比较之下，记分牌算法在发现冲突时，之后的冲突指令都会停顿，而 Tomasulo 算法不会停顿，因此记分牌算法的性能取决于程序本身的并行性。记分牌的调度是集中式的，而 Tomasulo 是由各 RS 分别控制，因此记分牌的控制逻辑更加复杂。

4 实验结果

基本测例见 `Log` 下的日志文件，两个性能测例的执行时间分别为：`Big_test.nel` 为 1.113s，`Mul.nel` 为 0.338s。我个人认为由于 RS 的数量有限，所以算法的实现方式不会对执行效率产生太大的影响，主要的时间都花在 I/O 上了。