Tomasulo 模拟器实验报告

计72 谢兴宇 2017011326

编译

本项目的源代码只有`tomasulo.cpp`和`tomasulo.h`两个文件，可以最基础的编译单文件 C++ 项目的方式来编译。

比如，在 Ubuntu 下，如果安装了 GCC，那么就可以直接：

```
g++ tomasulo.cpp -o tomasulo
```

运行

需要一个不带后缀的文件名作为参数，将自动从`./TestCase/`目录下寻找相应的`.nel`文件，并将其日志输出到`Log/`中相应的`.log`文件中。并会输出一个`output.md`，为 Markdown 格式的每个 cycle 的瞬时状态结果。

```
./tomasulo <filename> [-t]
```

例如

```
./tomasulo 0.basic
```

由于 IO 需要占用大量资源，所以我们提供了一个选项`-t`，以在性能测试时关闭对于每一个`cycle`的瞬时状态的过于大量的输出。

设计思路

所用的类的设计如下（形成多棵树形的继承关系）

- `Inst`：指令基类
  - `LoadInst`：LD指令
  - `ArithmeticInst`：算数指令
- `ReservationStation`：保留站
  - `ArithmeticBuffer`：算数指令保留站
  - `LoadBuffer`：加载指令保留站
- `FunctionUnit`：功能单元
- `Register`：寄存器

每一条指令会在三个地方出现：

1. 指令流
2. 保留站
3. 功能单元

简便起见，我们将所有的时间戳都记录在了`1`中，并维护了`3->2`和`2->1`的单向指针。

核心的思路是将每一个周期的工作划分为三部分：

- `try_write()`：尝试将指令结果写回寄存器，并更新保留站。
- `try_issue()`：尝试将指令发射至保留站中。
- `try_execute()`：尝试将指令将保留站中的指令移入功能单元，并执行处于功能单元中的指令。

  说明：由于旁路的存在，我们将`try_write`置于最前——而不是最后——以更简便地模拟真实硬件中发生的情况。

性能优化

在最初的实现中，为了方便起见我使用了大量 STL 中的数据结构。之后为了优化性能，我将它们改为了比较贴近底层的 C 风格的实现，运行效率有了极大提升，结果如下表所示。

测例	运行时间
Big_test	2.454s
Mul	0.016s

算法分析

Tomasulo 算法的特点是将状态记录分散到各个不同的部件中，这导致其细节相对繁杂。但基本思路清晰易懂，即用 reservation station 之间的依赖来简化寄存器之间的依赖，用 common data bus 来处理依赖关系。

Tomasulo 算法与记分牌算法的差异：

- 记分牌算法集中记录状态，而 Tomasulo 算法的状态是分布式记录的
- Tomasulo 算法支持寄存器重命名，一定程度上可以消除 WAR 和 WAW 造成的阻塞。