Tomasulo 实验报告

2016011370 蔡振廷

模拟器设计思路

主要的思路是在模拟硬件行为的同时方便UI的设计，因此我采用了类似MVP设计模式的方法，分离逻辑模块和数据模块。

模拟寄存器、保留站、运算单元的模块放在 `status.py` 中；tomasulo 模拟器的框架写在 `simulator.py` 中；操控 simulator 的核心算法逻辑写在 `controller.py` 中；UI 展示框架写在 `frontend.py` 中

具体的运行过程是将前置要求(nel代码路径、硬件数目等)输入到模拟器中，即可调用 `status` 建构符合要求的硬件模拟器。接着可以选择 `step` 或 `run` 进行单步或者完整运行 tomasulo 算法，模拟器会将当前的状态传给 `controller` 进行相应的运算和状态修改。以上就完成了命令行级别的 tomasulo 算法模拟器，但是为了方便手动调试和直观的得到结果，`frontend` 会提供格式化的输出信息并支持观看任意 cycle 的结果。

完成的功能

基本要求

1. 能够正确接受任意 NEL 汇编语言编写的指令序列作为输入

   > *注：输入的结尾不能有空行*

2. 能够正确输出每一条指令发射的时间周期、执行完成的时间周期、写回结果的时间周期

   > *注：命令行版本仅支持**最后一次执行的时间**；GUI版本支持显示最后一次执行的时间以及第一次执行的时间*

3. 能够正确输出各时间周期的寄存器状态

   > *注1：同上，GUI版本有完整支持*
   >
   > *注2：DIV命令支持的是标准python实现，可能会跟cpp实现有出入*

4. 能够正确输出各时间周期保留站状态、LoadBuffer状态和寄存器结果状态

   > *注：同上，GUI版本有完整支持*

扩展实验要求

1. 设计美观的交互界面

> *支持 `step`：单步调试*
> *支持 `run`：运行到结束*
> *支持 `first`：看第一次完成的周期*
> *支持跳转至任意 cycle 查看状态*

2. 实现高效的模拟算法

> *实现逻辑和 tomasulo 的硬件实现、人眼做题的实现接近，高效的同时保证直观性*

运行方式

1. 命令行版本

> *运行 `python3 simulator.py` 即可显示 `test0.nel` 的模拟结果。*
> *修改 `main()` 中的 `init()` 可以更改保留站、运算单元、寄存器的数量*
> *修改 `main()` 中的 `cycle` 可以改变运行的周期*

2. GUI版本

> *运行 `python3 frontend.py`*
> *输入 NEL 代码的路径*

样例程序

样例程序为课堂 PPT 上解说的那个例子，结果正确

样例路径为 `test3.nel`