Tomasulo 实验

计01　容逸朗　2020010869

已实现的功能

- 必做功能：Tomasulo 算法
- 选做功能：分支预测算法

实现内容

1. 处理前端流出的指令 (Backend::dispatchInstruction)

- 首先利用 `getFUType` 取得指令对应流水线部件的名称，然后将指令直接插入 ROB；
- 对于 `FUType::NONE` 类型的指令不需要进一步操作，返回 True 便可；
- 对于其他指令，先检查对应部件保留站是否还有空位，若无空位则返回 False，暂停前端流水；
- 然后调用保留站的 `insertInstruction` 函数插入指令，再把 RD 对应的寄存器设为 Busy 并记录当前的 ROB Index，最后返回插入成功（True）。

2. 指令提交 (Backend::commitInstruction)

- 首先检查指令是否为 `EXTRA::EXIT`，若是则返回 True，退出程序；
- **高级功能**：检查指令是否为六种条件跳转指令之一，若有则需要把 ROB 中的 pc 项以及当前分支跳转情况（是否跳转、成功跳转位置）通过前端的 `bpuBackendUpdate` 插口返回；
- 接下来，先用 `getPopPtr` 取得当前指令的 ROB Index；
- 然后判断指令是否为三种存储指令（SB/SH/SW），若是则需要在 StoreBuffer 中找到对应地址（此处是 `entry.state.result`）的值并写回 `data` 中；
- 否则，若为一般指令则需要把指令计算好的值（`entry.state.result`）写回寄存器堆中；
- 紧接在 ROB 中 pop 掉一个表项，表示 commit 成功；
- 最后检查分支跳转是否预测失败，若失败则根据 `entry.state.actualTaken` 判断前端要跳回的值，jumpTarget 或 pc + 4，同时利用 `Backend::flush()` 清除后端各部件的内容。

3. 保留站插入指令 (ReservationStation::insertInstruction)

- 按照算法，先找到空闲槽位；
- 然后判断 Rs1 (Rs2) 在寄存器堆的状态，若不是 Busy 则从中取值即可；
- 若为 Busy，则先找到对应寄存器数据所在的 ROB Index，然后查看对应表项是否准备好；
- 若准备好，则从中取值并写入保留站，否则置 Rs1 的待唤醒项为 True，等待对应项完成；
- 然后把 Rd，当前指令 ROB Index 放入此保留站槽位中，并记此槽位为 Busy。

4. 保留站唤醒 (ReservationStation::wakeup)

- 遍历所有 Busy 的槽位，然后查看 Rs1 (Rs2) 是否处于待唤醒状态，再比对 ROB Index 项；
- 若匹配，则把值写入槽位并关闭寄存器的待唤醒状态。

5. 保留站是否可发射 (ReservationStation::canIssue)

- 遍历所有 Busy 的槽位，查看 Rs1 (Rs2) 是否处于待唤醒状态，若都不是则有槽位可发射。

6. 保留站发射 (ReservationStation::issue)

- 找到任意一个可发射槽位，然后把对应槽发射出去，最后置当前的槽为空。（去掉 busy 标志）

选做功能

本次实验中，我还实现了分支预测算法，对代码了作出如下改动：

1. 设计方案

- **BTB：** BTB 的大小为 4096 项，每个表项有 valid, predict, index 和 target 四项，分别表示表项是否可用，当前预测状态，BTB 比对项和目标分支项。
- **分支预测：** 修改前端的 `calculateNextPC` 和 `bpuFrontendUpdate` 函数，需要从根据当前 pc 的 [13:2] 位找到 BTB Offset，然后检查表项是否可用且 BTB Index 是否和 pc 的 [31:14] 位相同，符合者可以根据 `predict` 的值是否大于等于 2 来判断是否跳转；
- **分支预测器更新：** 和分支预测时一样，不过当表项不同时需要更换内容为当前指令，并根据跳转情况置 `predict` 为 3（成功）、0（失败）。若表项相同，则根据跳转情况置为 3（成功、原表项非 0）、2（失败、原表项为 3）、1（成功、原表项为 0）、0（失败、原表项非 3），以此表示分支的跳转情况。

总结

- Commit ID：`051489eb2c6206fe12c3e898b0012f96150306b5`（位于 `distribute` 分支）