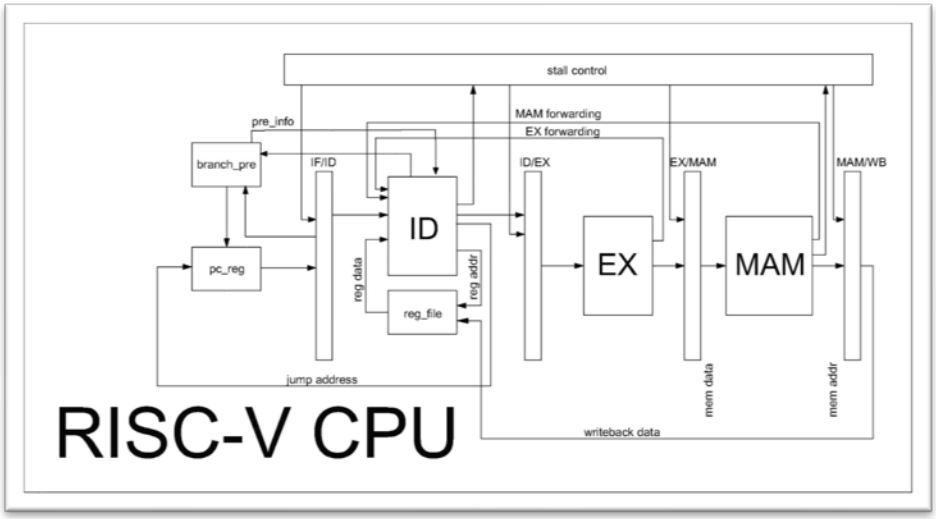
RISCV – CPU

16 ACM 李怿铠

处理器架构

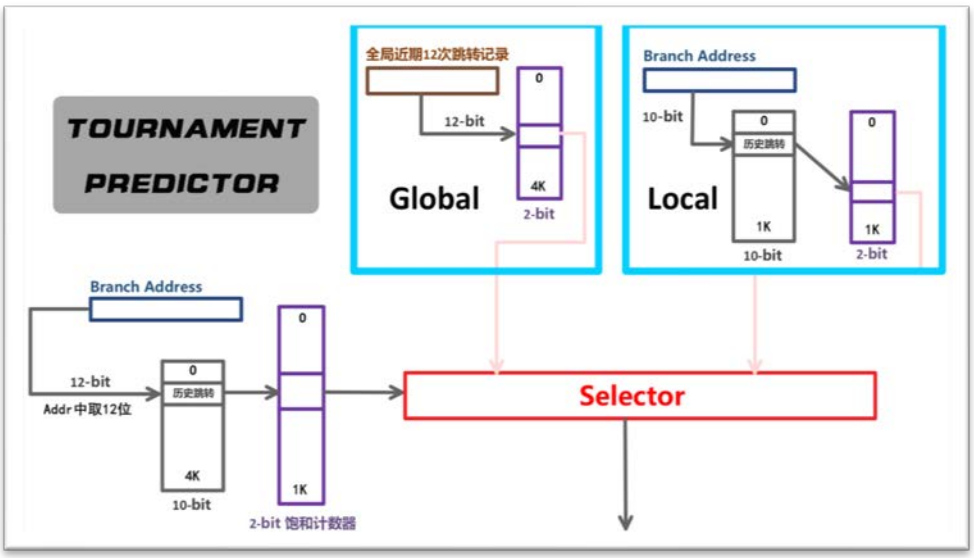


架构如图所示，在实现正常五级流水的基础上添加 tournament 分支预测。

架构实现“个性化”

- 在 ID 阶段依据 opcode、funct3/7 将指令映射至 OpenMIPS 的八位操作码，继承其后续实现
- 对于跳转指令在 ID 阶段解析出之后，立即计算出跳转地址修改 PC 寄存器，将跳转指令的损失降至一周期
- 将 LUI、AUIPC 定义为跳转类型指令，ID 阶段解析时直接将结果存为立即数，后随流水流入寄存器
- ID 阶段解码时将立即数存储进 rd2 寄存器，合并 EX 阶段运算分支中的与立即数相关操作

特色功能 —— TOURNAMENT 分支预测



功能实现

tournament 分支预测模块分为两个部分：**跳转预测** & **饱和计数器更新**

跳转预测

a) 由 inst_rom 导入 pc、inst 触发，检测若为跳转指令则进行预测
将预测结果告知 PC 同时随指令流入 ID 阶段保存

b) ID 阶段在下一周期指令到达时，判断预测是否正确。
若错误，将正确结果发送给 PC，同时将 IF/ID 阶段 stall

状态更新

ID 阶段为跳转指令时触发

根据 ID 发回的保存结果与预测正确性判定，更新饱和计数器状态

注：考虑到 pc 从零开始每次递增 4，且长度为 32bit，取 inst[13:2]进行映射。

心得体会

参考书目**《自己动手写 CPU》**

书本首先进行指令集架构和 Verilog HDL 的基础知识讲解，然后采用步步扩充的方式搭建出完整的处理器，依据书本中实现模改即可方便搭建自己的处理器。

其中 **OpenMIPS** 与 **riscv-cpu** 的区别以及一些注意点：

- riscv 中没有乘除法运算，因此不需要相应的乘除法模块与 hi，lo 寄存器
- riscv 对于指令中立即数生成规则有专门说明，参见文档 riscv-spec-v2.2（后找到中文版）
- riscv 中 rd/rd1/rd2 位置相对固定，可简化 ID 阶段解码实现
- riscv 不再使用延迟槽机制

发现问题

1) 在与 OpenMIPS 的 8 位操作码对接时，发现其指令总数远远不及 $2^8 = 256$，又删去了大量非 riscv 指令之后，思考是否可以通过哈夫曼树方法进行操作码的重新分配？
后回忆起 align 指令，担心在数据传输过程中会有对齐问题，于是放弃修改。
2) 由于 riscv 中的跳转是依据 PC 或 reg_data 的偏移来进行，需要进行加法操作。
若由 EX 阶段的 ALU 来运算，则为一条跳转指令需要 stall 两个周期，浪费过多。
若选择在 id 阶段做完，则又似乎不需要 EX 阶段变为四级流水。
3) 书本中实现的 memory 为哈佛结构，即 inst 与 data 分立存放，不支持 selfmodify 同时若需要与电脑通信，也需要将 inst_rom 与 data_ram 合并。

特别鸣谢：

范舟与**陈竞潇**同学提供的测试数据与**李沐阳**同学在架构实现上的指导。