ICS-Pre 讲稿

Part I 快速开发指南

如何快速开发一个编程错误少、代码风格清晰、易于使用和维护、自然实现硬件电路属性的后端？

Point.1 硬件设计风格

- 这是我们的**中心原则，一切以硬件视角进行模拟**，即可利用硬件本身的现实属性自然地实现相关现象
- 包含许多自定义硬件类型
- 有两个重要硬件问题需要解决：
    - 上升沿更新
        - 阶段寄存器中变量为寄存器类型、寄存器类型变量**必须**把值传给对应线路类型变量，对应线路变量再赋值给下一阶段寄存器
    - 硬件电路并行性
        - 简单分析各个组合逻辑块和阶段的依赖关系，按**逆序**依次执行
- 每个组合逻辑块写为一个返回值 void 的函数，输入变量传值，输出变量传引用，内存等数组不做区分传指针
- 每个阶段寄存器使用 namespace 划分

Point.2 多层次结构

- 定义层
    - 宏定义、类型定义
    - 工具箱
        - 标号转化名称
    - 硬件组件层
        - 定义了所有逻辑块的底层实现
    - 硬件定义层
        - 定义了所有系统硬件，相关寄存器和线路变量
- 控制层
    - 阶段控制层
        - 每个阶段统一分为初始化、运算和更新三步
    - CPU 整体控制层
        - 定义了 CPU 初始化、运算和更新方法（所有阶段合在一起控制）
- 用户层
    - 模拟器初始化层
        - 编写了从 .yo 文件中读取内存字节的函数
    - 单元测试层

            - 对特定函数功能进行正确性测试
        - Y86 用户层
            - 控制整个 Y86 模拟器的初始化、运行和终止行为
- 主函数调用层（终端层）
    - 可以选择进行测试还是进行模拟

Point.3 大量的预定义

- 宏定义（一共 96 有效行）
    - 最大内存范围、最大寄存器数量、PC 起始位，3 行
    - 标准程序状态码以及补充的程序状态：气泡对应的标号，5 行
    - 所有指令的字节块、代码部分以及功能部分对应的标号，72 行
    - 所有寄存器对应的标号，16 行
- 所有硬件模拟所需的类型定义
    - `mem_t` 内存单元类型 unsigned char
    - `reg_t` 寄存器类型 unsigned long long
    - `wire_t` 线路类型 unsigned long long
    - `ptr_t` 指针类型 signed long long
    - `stat_t` 状态指示器类型 unsigned short int
    - `cc_t` 条件码类型 bool
    - `imm_t` 立即数类型 unsiged long long
    - `cpu_t` CPU控制层类型 unsigned long long
    - `addr_t` 地址类型
    - ...

Point.4 周到的编程细节考虑

- 把暂停、气泡等放在阶段寄存器的更新步，正常把线路赋给寄存器对应正常更新，寄存器变量初始化对应加气泡，什么都不做对应暂停
- 所有的硬件定义都在组件（函数）定义之下，防止函数越过参数表访问不该访问的硬件变量
- 开头字母小写表示线路变量，大写表示寄存器变量
- 每个 namespace 中的非主函数仅严格调用下一层的函数，不再继续向下，主函数仅调用同 namespace 的其他非主函数，由此实现抽象级别
- 层次化很好地简化了代码。很多函数中仅有几行，比如次高层次的用户层的 Y86 运行函数，仅包含 Y86 init, run, end 三个函数调用，且没有任何参数。最高层次的主函数仅有一行，就是调用 Y86 运行函数
- 按标号规则定义名称字符串数组，标号即是对应名称的所在下标，由此便于实现标号到名称的转换

Point.5 其他

- 因为本身就是严格对应真实的电路图和硬件风格编程，因此完全不用担心模拟器是否对应真实硬件的问题

Part II 收获与体会

- 从这次 PJ 中，我们获得了：
    - 首次前端开发经验、前后端交互经验
    - 认识到了各个模块版本不匹配、不兼容的问题及解决方法
    - 熟悉了 html, js, css 的语法
    - 增强了对前端框架作用的理解和直观认识
    - 深刻认识了 Y86 模拟器的底层原理，以及更加标准的开发规范带来的便利