第一次实验报告

20307130112 马成

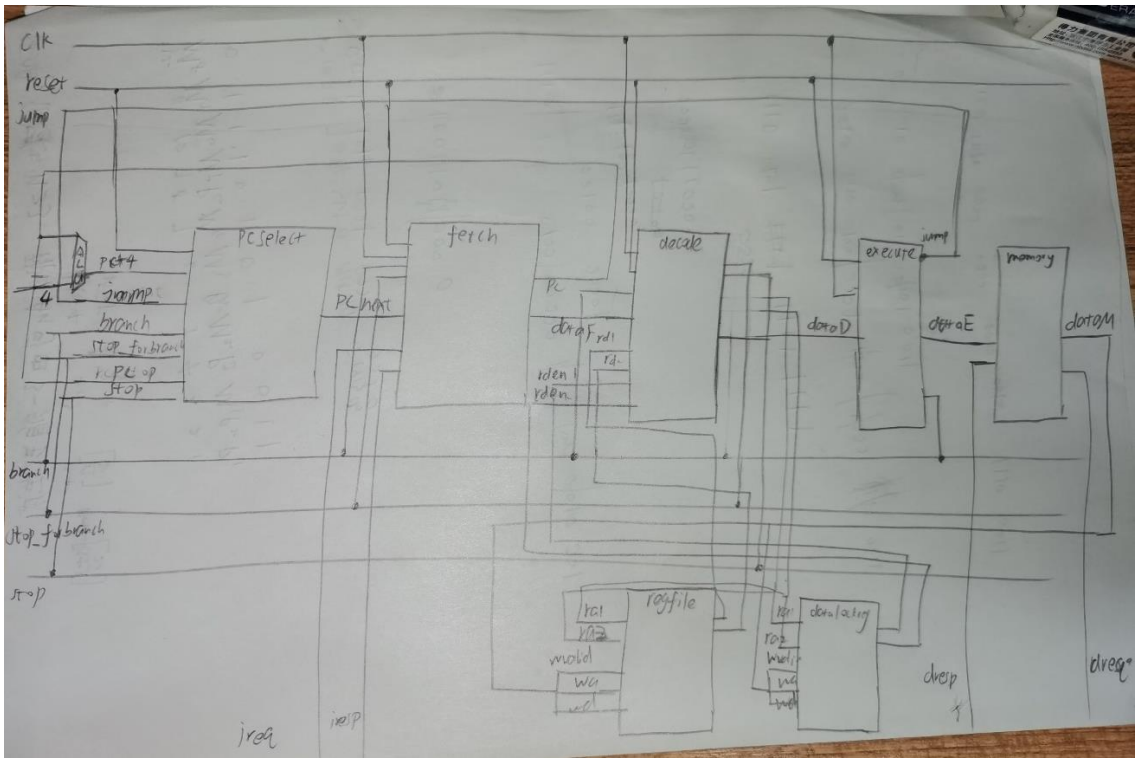
一、 解决冲突的策略

本次实验中我处理冲突的策略是使用气泡法。

1. 结构冒险：在这次实验中老师已经给我们搭建好了读取 pc 的模块和读写内存的模块，我只需要使用老师给出的接口即可。在这个实验中 pc 使用的模块和内存的模块分别用了两个 ram 不存在结构冒险的问题
2. 数据冒险：当发现有一个寄存器的数据即将被读取，但是检测到这个数据正在被某一条指令写入且还没有完成写入的时候。如果还从这个内存中读取没有写入的数据就会导致错误。因此在遇到这种问题的时候将 fetch 中的 pc 和 decode 中的 pc 暂时固定，并将他们输出的信号 dataD.bubble 置为 1 表示这是一个气泡，后面无须特殊的操作。在检测寄存器是否正在被写入的策略如下。我设置了两个类似于寄存器的 regfile，并将他们初始值全部置为 0。其中一个 regfile 纪录对应的寄存器地址中有多少次在 decode 的时候被标记请求写入。另一个 regfile 纪录对应的寄存器地址中有多次的 dataM 中被标记写入。可以看到，如果这两个 regfile 中给出了相同的数值则表示对应的寄存器地址未处在正在写入的状态，可以正常读取其中的数据。否则表示寄存器正在被写入，当前的 pc 需要暂时保存，直到 dataM 成功写入了对应的寄存器值，两个 regfile 中的值重新相同时才能继续流水线。
3. 控制冒险：当 decode 中发现了一个跳转指令（认为 beq 也是跳转指令，只是可能跳转到 pc+4 位置）应该暂时停止 fetch 读取指令，直到 excute 中计算出正确的跳转位置在继续读取。这样的操作是如果 decode 中检测到跳转指令（jal，jalr，beq）先将 stop_forbranch 写为 1，表示检测到一条跳转指令。在 excute 模块中给出一个 branch 数据，将这个数据写为 1 表示跳转指令计算完成（在 excute 接收到非跳转指令的时候默认将 branch 写为 0）并且通过 jump 给出即将跳转的位置。在 pc 中如果 stop_forbranch==1 && branch==0 表示跳转位置没有确定，继续等待，如果 stop_forbrach==1 && branch==1 表示可以进行跳转，将 pc 写为 jump 即可。
4. 补充，如果数据冒险和控制冒险同时出现，会先处理数据冒险否则不能保证在 jump 的计算中使用了尚未完成写入的寄存器中的操作。实际上如果两个冒险同时出现，数据冒险会优先将 decode 中的 dataD 输出接口的 bubble 写为 1，这样 excute 接受到这个信号的时候由于检测到这是一个气泡不会进行任何操作，所以修改 branch 即不会停止 pc 因为跳转指令等待的状态。

二、 简易电路图

1. core 的电路图

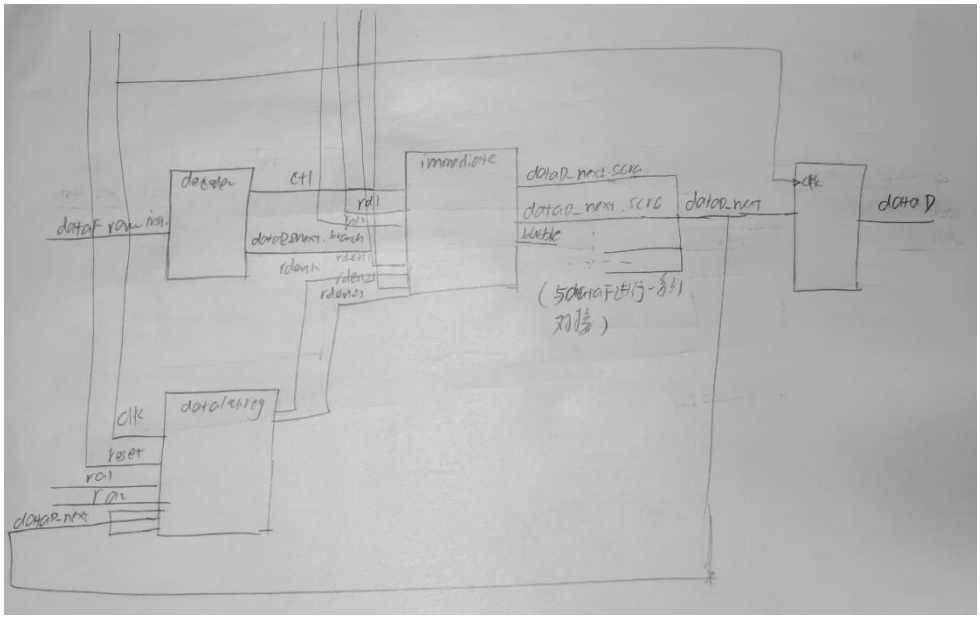


Pcselect 用于根据情况选择 pc_next

Fetch 用于根据 pc_next 读取指令并给出一个封装过的 dataF

Memory 用于根据 ctl 是否为访存指令来操作内存单元

2. decode 电路

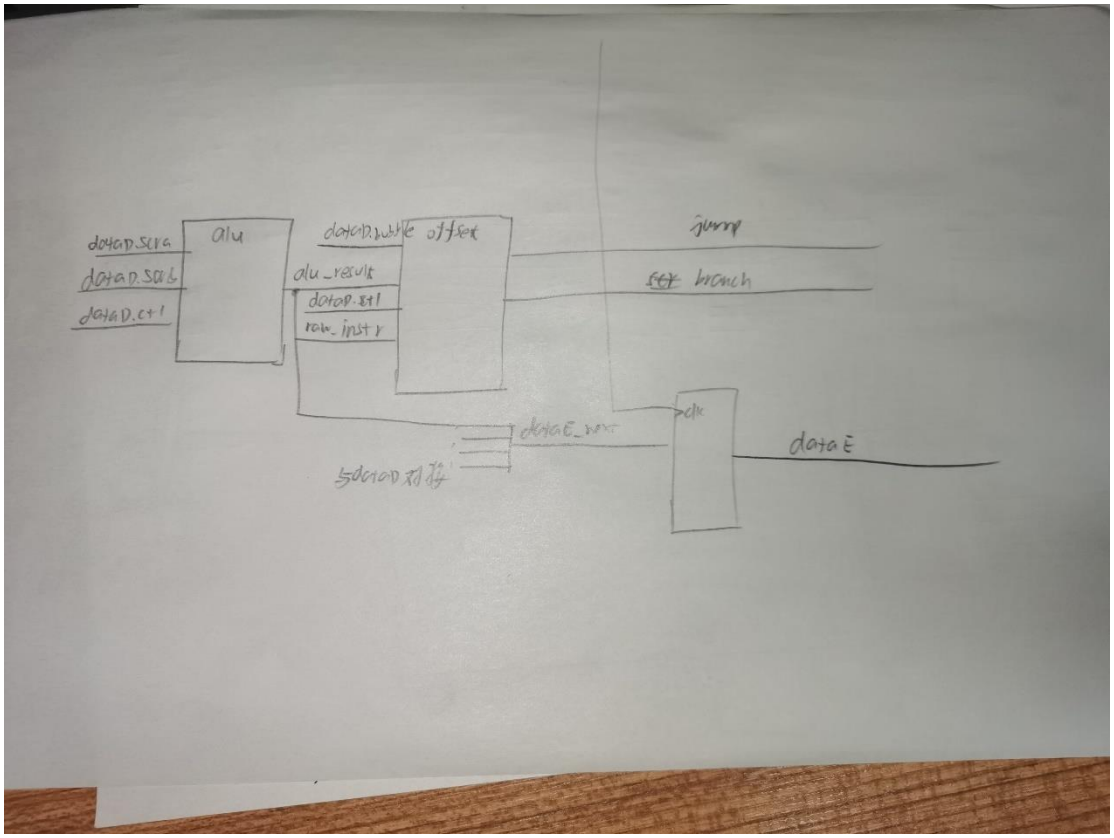


Decoder 用于分析指令给出指令的类型

Immediate 用于给出 alu 的操作数

Datalock 纪录即将写入的寄存器地址和得到即将被读取的寄存器地址的状态，用于判断是否需要气泡

3. execute 模块



Alu 用于计算 dataM 的值（有时求出的是需要访存的内存的地址）
Offset 用于计算即将跳转的 pc 的值，同时输出 branch 信号用于提示 pcselect 模块