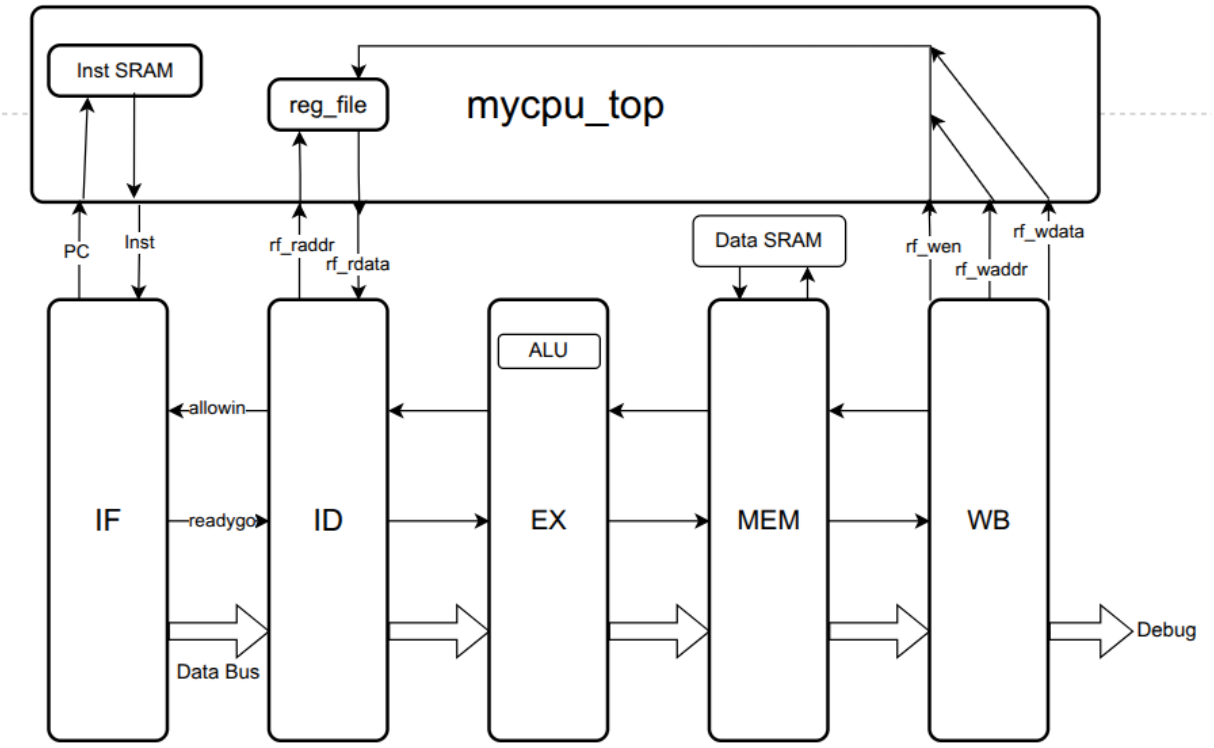
Project 2 实验报告

石曜铭 2023K8009929037

10/14/2025

1 处理器结构设计图



2 主要设计点

2.1 握手信号设计

- `valid`: 在每个模块内部维护，表示该模块该周期是否有效，即未被 flush 掉。
- `readygo`: 拉高表示该模块已经做好准备将数据传到下个阶段。
- `allowin`: 拉高表示该模块下个周期可以接受来自上个阶段的数据。

`allowin` 表达式：`allowin` = (!`valid`)|(`readygo`&`next_allowin`) 。这里第一项表示如果当前状态无效，那么可以接受上个阶段的数据以挤掉气泡；第二项表示如果该阶段数据即将进入下个阶段，那么也可以接受上个阶段的数据。

2.2 与内存交互的时序设计

本实验中的 SRAM 是同步写同步读的，意味着与内存的交互都需要经过一个周期才会真正结束。具体体现在取指阶段和 load 指令，都需要等待一个周期才能得到结果。虽然可以通过将数据接收放在下个阶段来解决，但这样难以扩展到多周期等待的内存交互。

于是，对于 load 指令，我选择模拟一个内存握手信号，恒置为 1，这样就会自然有一个周期的等待。

在 IF 阶段，因为指令是同步读的，所以每次输出的 PC 实际上是 nextPC，这样下个周期自然得到了与当前真正 PC 对应的指令。

2.3 数据冲突：数据前递

数据冲突本质上是当前指令是寄存器写，在还没有到 WB 阶段时，另一条指令的 ID 阶段恰好需要读这个寄存器。

数据前递用来解决写寄存器的数据在 EX 产生的情况。只需要分别在 EX 阶段和 MEM 阶段输出已经确定的寄存器写的值和地址，接到 ID 阶段，那么在 ID 阶段就可以直接得到前两条指令的寄存器写信息，跟当前需要读的寄存器做比较，讨论情况即可。

寄存器的代码也要做一个改动，即如果异步读的地址恰好等于写地址，就直接把写数据返回。这样就可以对每条指令只往后考虑两条指令的数据前递。

2.4 数据冲突：Load-Use 阻塞

如果写寄存器的数据在 MEM 阶段才产生，那么就无法通过数据前递来解决冲突。在 ID 阶段，当检测到一条指令的上一条指令是 load，且写寄存器的地址恰好被读，那么需要阻塞该阶段直到上一条指令的 MEM 阶段完成，并再开一个信号用来传递内存读出的数据。

2.5 控制冒险

如果一条指令在 ID 阶段被发现不应该执行，那么应该把 `valid` 置为 0，表示该指令的数据失效。相应地，为了保证被失效的指令不会对控制信号产生影响，应该把所有的控制信号都和 `valid` 取与，这样可以避免控制冒险，例如避免被跳过的 branch 指令还会影响 PC 跳转的情况。