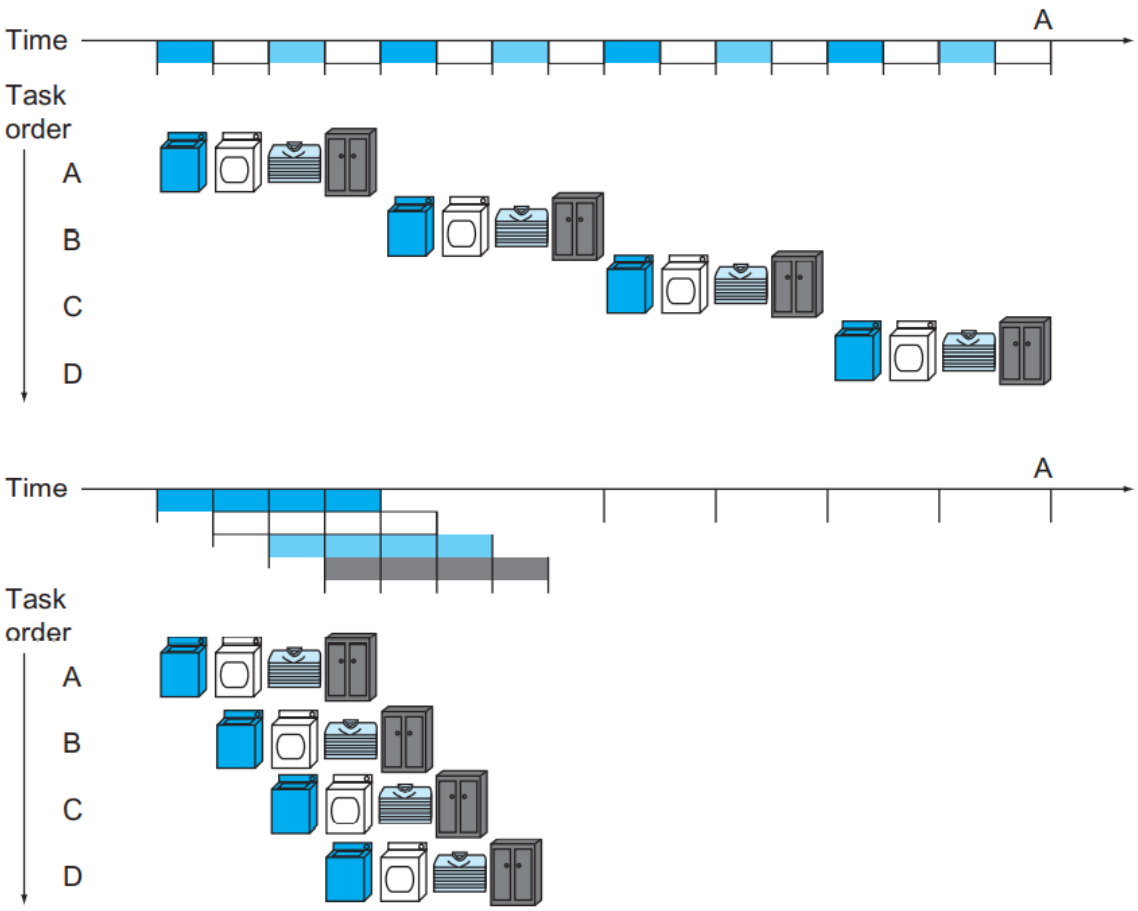
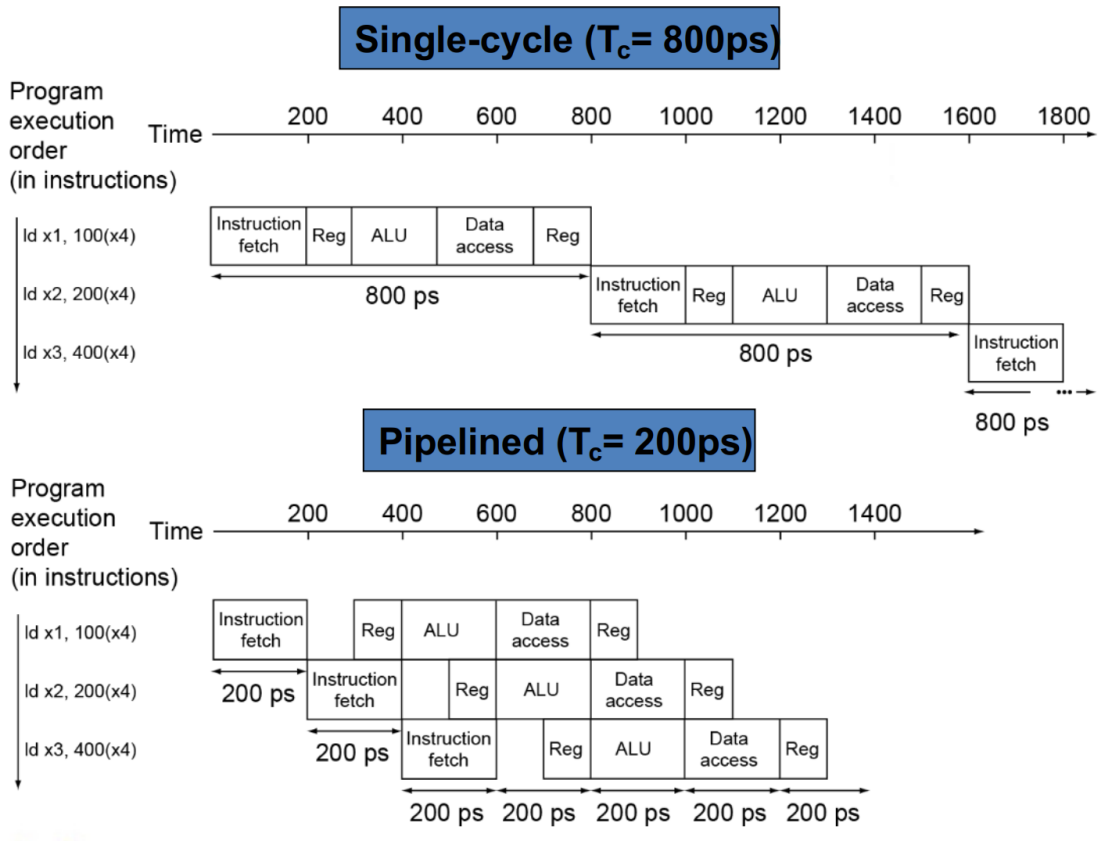
chapter4-2 流水线CPU



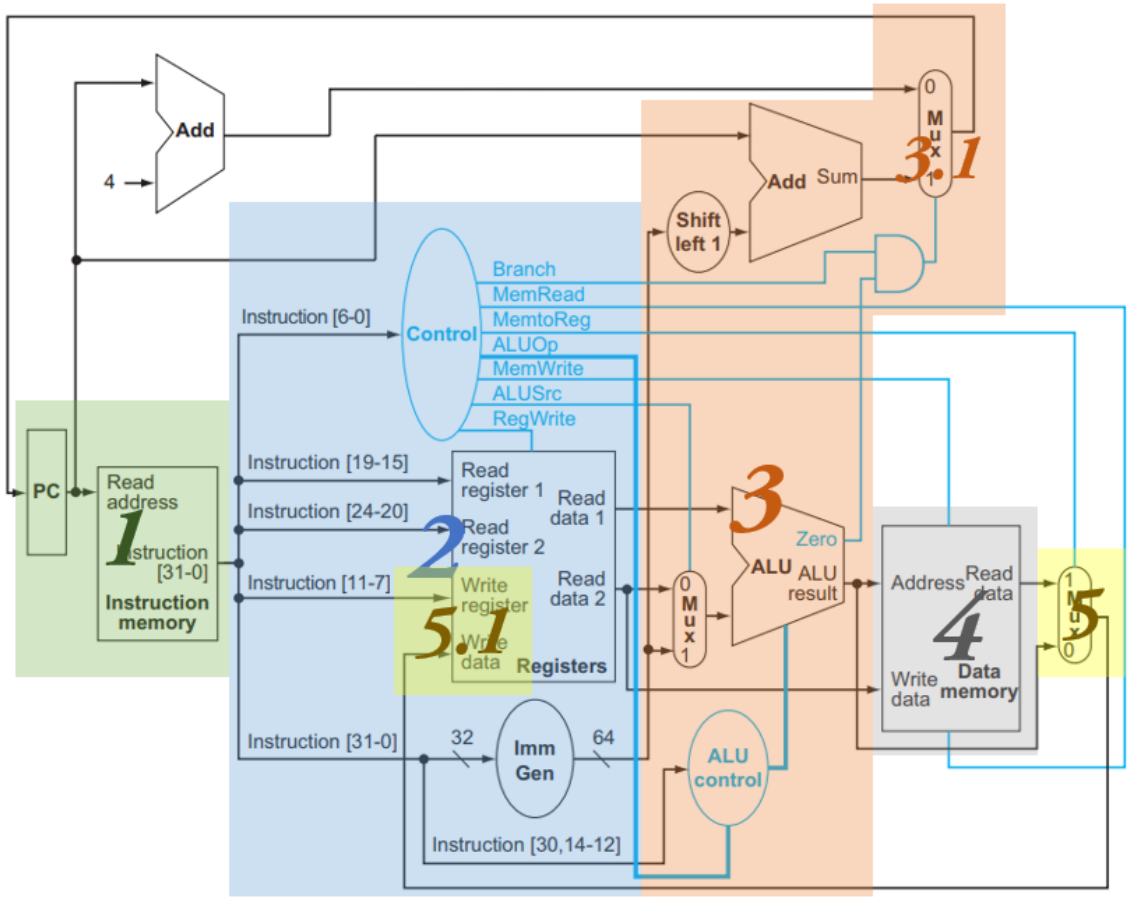


这样时钟周期只取决于最长的那个步骤

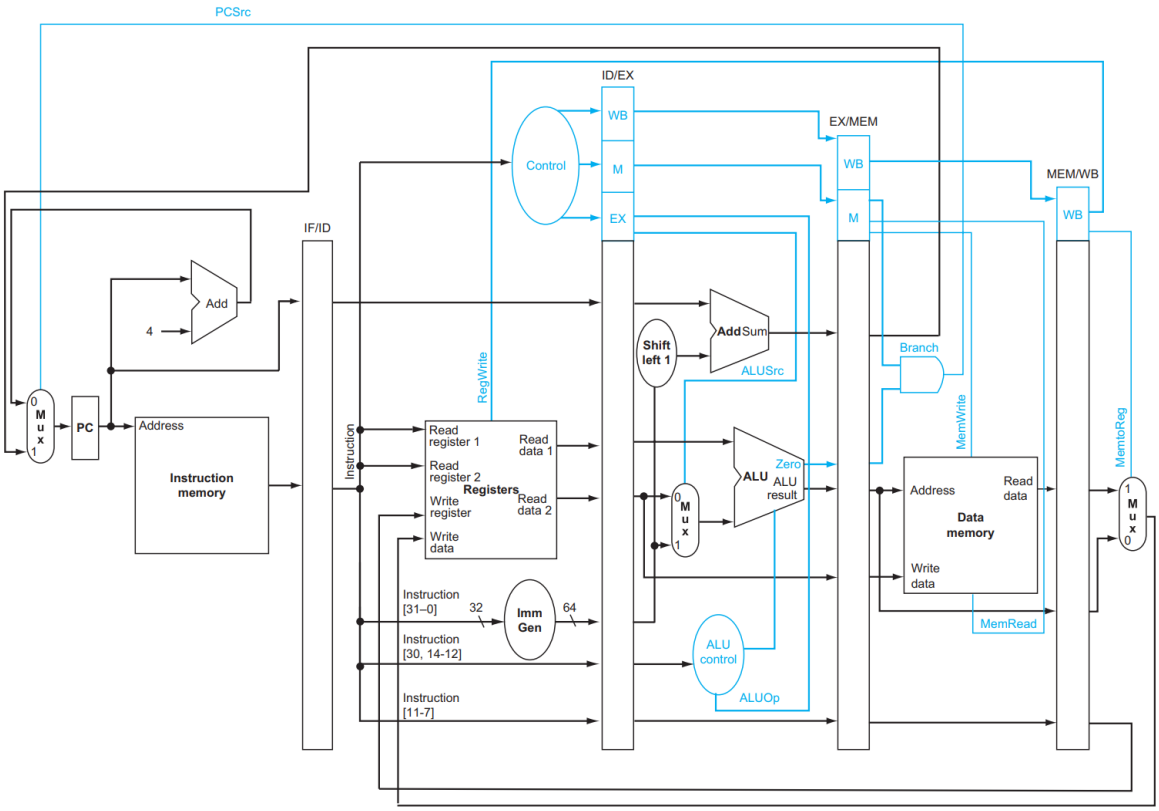
中间寄存器

分为五个阶段

1. **IF, Inst Fetch**，从内存中获取指令
2. **ID, Inst Decode**，读取寄存器、指令译码
3. **EX, Execute**，计算操作结果和/或地址
4. **MEM, Memory**，内存存取（如果需要的话）
5. **WB, Write Back**，将结果写回寄存器（如果需要的话）



当然，实际上大家务必注意更新 PC 的步骤放在哪里，无论什么时候提前 PC 计算都能优化性能，考试中常常放在 ID 或者 EX。



结构冒险（Structure hazards）及其优化

1. 要**在下降沿写入** memory 或者寄存器堆，实现**先写后读**,可以使得寄存器只需要考虑两条指令前递，memory只要考虑一次前递
2. 由于在流水线中 IF 和 MEM 都有可能用到内存，为了不出现结构竞争，我们将数据和指令放在不同的地方，**指令数据分开**以避免同时出现要访问内存而导致的结构竞争。
3. 这种**备份硬件**思路也可以用在其他的元件上，例如增加浮点加法器数量
4. 给浮点加法器加入流水线
5. stall |= structural hazard
6. 当然我们已经默认实现了**多个输入输出端口**的操作，例如寄存器堆的实现

数据冒险与前递

数据冒险

定义

在顺序流水线中，只有 RAW 会造成数据冒险，但是乱序流水线中，WAW 和 WAR 都会造成数据冒险，RAR 则无论如何都不会造成数据冒险。

解决方案

1. stall |= (rd(EX/MEM) == rs1/rs2 (ID/EX))|| (rd(MEM/WB) == rs1/rs2 (ID/EX))
2. Compiler scheduling，即通过编译器调整汇编指令顺序
3. Forwarding
4. Double bump：通过器件解决 load-use hazard

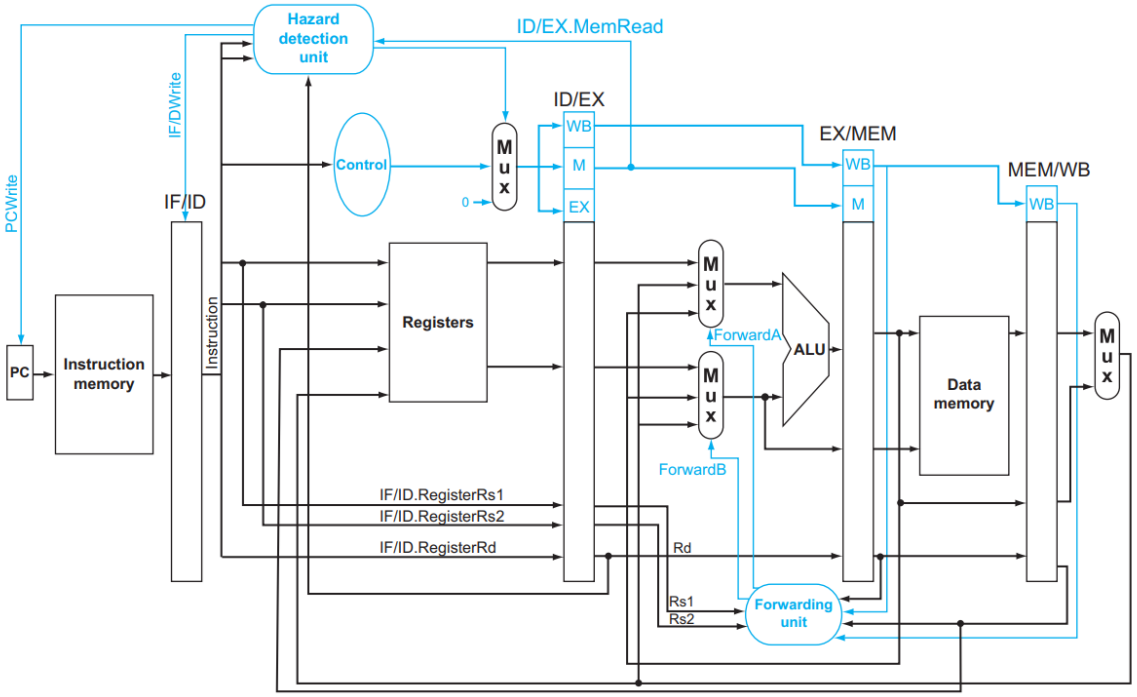
前递

控制信号

1. 出现以下情况时，即之前的计算结果或者存储结果需要**使用**(注意，Rd $\neq 0$，且 RegWrite 有使能)
    - EX hazard
        - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs1))
            - ForwardA= 10
        - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs2))
            - ForwardB = 10
    - MEM hazard
        - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs1))
            - ForwardA= 01
        - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs2))
            - ForwardB = 01
2. 出现以下情况时，即出现 load-use harzard，需要 stall 一个周期
    - ID/EX.MemRead and ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or (ID/EX.RegisterRd = IF/ID.RegisterRs2))
3. 如果需要储存计算出的结果
    - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
        - and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
            - and (EX/MEM.RegisterRd = ID/EX.RegisterRs1))
        - and (MEM/WB.RegisterRd = ID/EX.RegisterRs1))
        - ForwardA = 01
    - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
        - and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
            - and (EX/MEM.RegisterRd = ID/EX.RegisterRs2))
        - and (MEM/WB.RegisterRd = ID/EX.RegisterRs2))
        - ForwardB = 01

数据传递

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.



控制冒险（Control Hazards）与分支预测

控制冒险

当处理器遇到分支指令（如if-else、循环）时，它需要决定下一条指令是顺序执行还是跳转到目标地址。在做出这个决定之前，下一条指令的地址是未知的

最简单的解决方法是 stall，即直接停止取值，当然在什么位置计算出结果决定了要 stall 几个时间段

- （MIPS）中，分支条件在**EX**阶段计算branch结果，到算完的时候 IF 和 ID 被闲置，已经浪费了两个周期，算完之后还要一个周期算PC，又浪费一个周期，总共浪费三个周期
- 但是书上一些例题中在 ID 阶段计算出 branch 结果，因此只浪费两个周期

当然有一些其他方法（Reducing Branch Delay），例如在 ID 阶段就算出来分支条件将错误取指的指令清除，一点也不会浪费，除非这也有load-use hazard。

也可以使用编译器调整指令顺序。

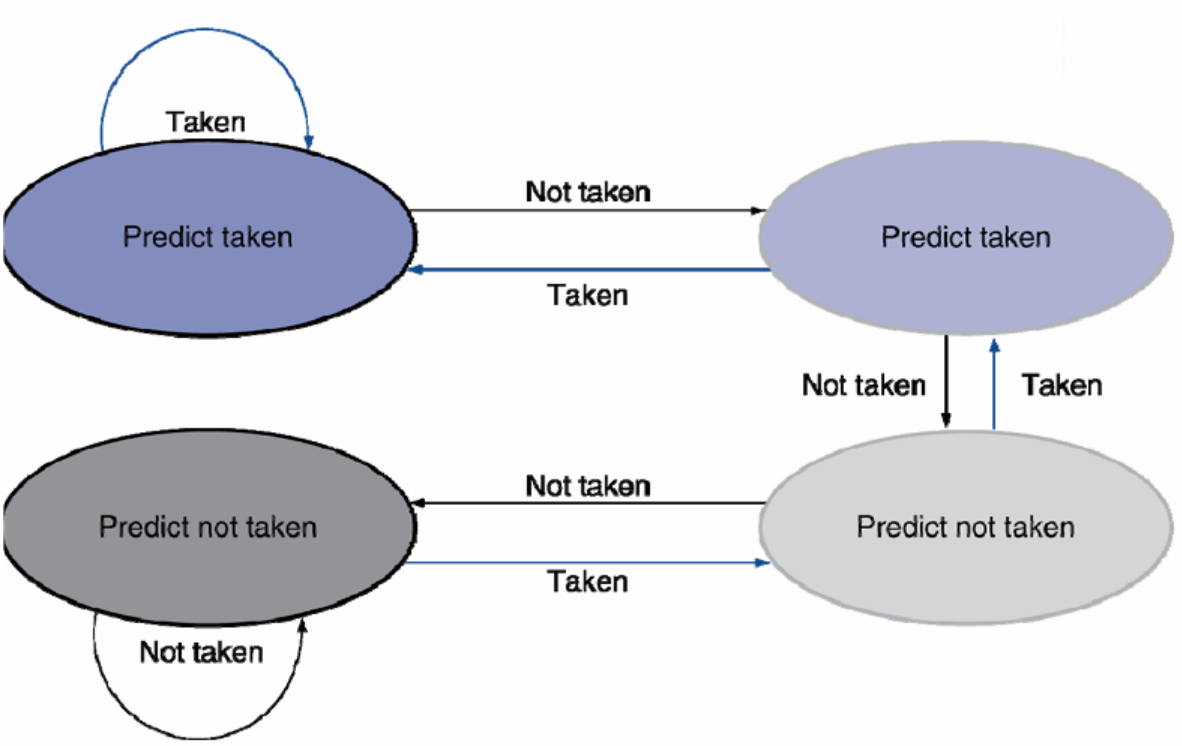
分支预测（Prediction）

我们会使用分支预测的方法，分支预测最简单的就是一直预测不跳转（ Predict-untaken），真要跳转了清空就行

当然进阶一点就是记录上次跳转的位置之后记录（Predict-taken ）

更进阶一点是采用以下方法动态预测是不是跳转（Dynamic Branch Prediction）

Branch History Table: Lower bits of PC address index table of 1-bit values



杂题选讲

1. 如何调整指令顺序解决问题