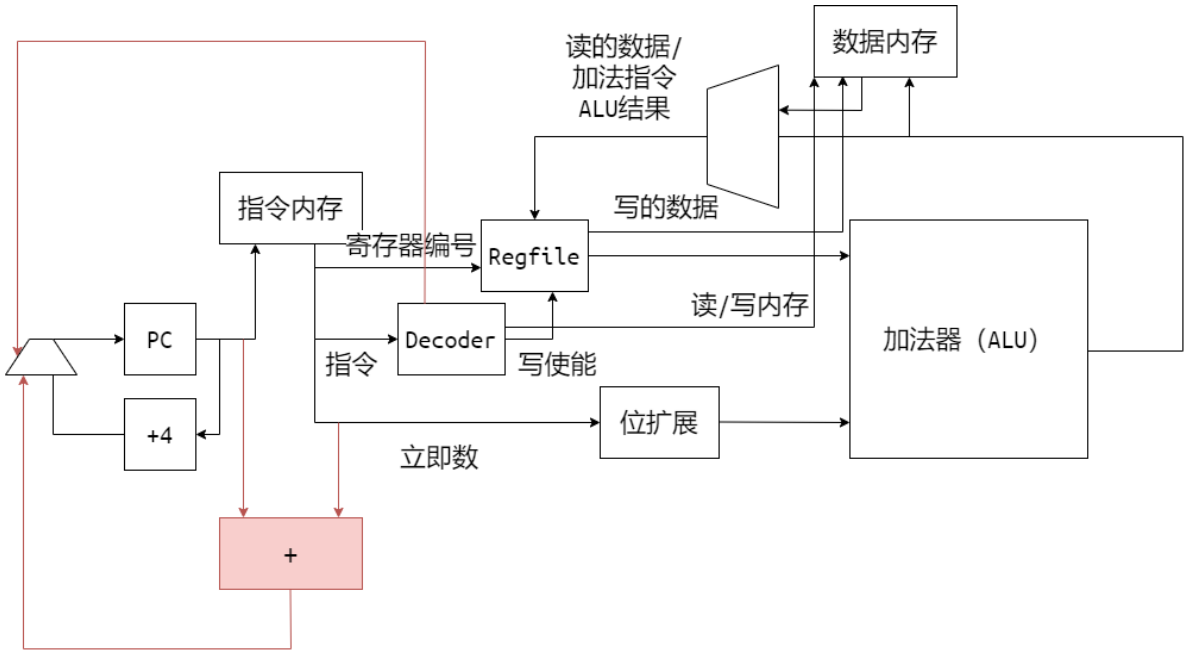
五级流水线

洞察 insight

回顾单周期数据路径：



以下几个组合逻辑部件的延迟较高：

- 读指令内存（4096选1，32位宽）
- 读寄存器（32选1，64位宽）
- ALU（约16选1，64位宽；另有加法进位链）
- 读数据内存（256选1，64位宽）

单周期的**关键路径**是这几个部件的延迟之和。

> 关键路径是电路中组合逻辑延迟最高的路径，限制了电路的时钟频率。时钟周期为定值。

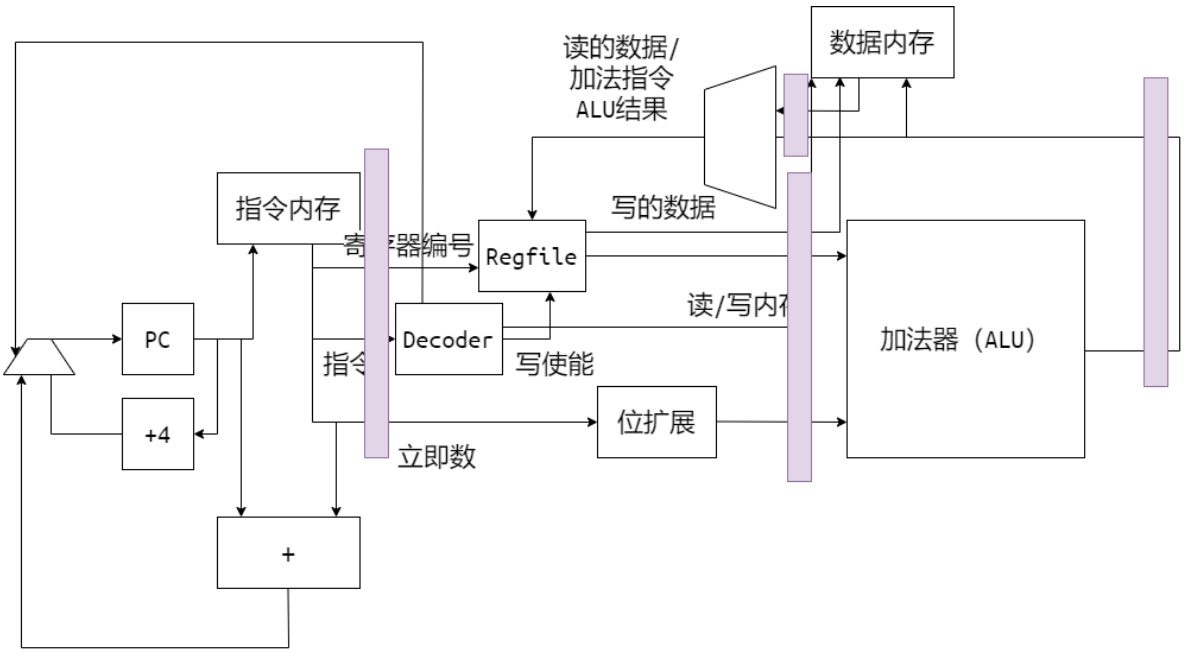
流水线可以将关键路径延迟降低到这几个部件的 max。

设计流水线

流水线的原理与原则：

- 所有组合逻辑模块的输出仅取决于输入。这个输入是来自其他组合逻辑模块，还是时序逻辑，并不重要
- 理想情况下，j 级单发射流水线执行 n 条指令需要 $(n+j)$ 个时钟周期，平均每条指令执行 $(n+j)/n$个时钟周期
- 修改全局状态的行为与单周期处理器完全一致

简单地做下切割：



也可以参考这张图：https://fducslg.github.io/ICS-2021Spring-FDU/arch-lab/lab1.html#12-%E4%BA%94%E7%BA%A7%E6%B5%81%E6%B0%B4%E7%BA%BF

紫色框里存后面流水线需要用到的，不用存后面流水线生成的。

流水线冲突

流水线带来的问题是，一条指令的结果可能要晚若干个周期得出。下面将介绍一些冲突场景，以及用 stall, bubble, forward 的方法来解决它们。

跳转

Decode 阶段才识别出这是一条跳转指令、算出跳转地址、得出是否跳转；可这个时钟周期里， Fetch 阶段做什么呢?

我们采用默认不跳转的方法，Fetch 取顺序执行的下一条指令。如果不跳转，则正常运行；如果跳转，则顺序执行的下一条指令不应执行，用气泡 no operation(nop) 替换这条指令。提示：设计 `control_t` ，它为全0时不改变任何全局状态，即为 nop。

```
always_ff @(posedge clk) begin
    if (jump) instr <= '0; // Warn: this is an invalid instruction
    else instr <= instr_nxt;
end
```

数据冲突

第 i 条指令的结果会在 Writeback 阶段写入寄存器。而 $i+1$ 条指令会在 Decode 阶段读取寄存器。如果不加以处理， $i+1$ 条指令读到的数据就是错误的。

我们可以采用 stall 阻塞的方法解决：

- 让前面若干级流水线寄存器不更新值
- 让后面若干及流水线继续流动

这样，过几个时钟周期， $i+1$ 条指令就能从寄存器读到正确的值了。

我们不难发现两点：

- 某级流水线寄存器阻塞，前面的流水线寄存器似乎也需要阻塞，否则就消失了

- 某级流水线寄存器阻塞，后一级流水线寄存器似乎需要清空，否则这条指令将多次改变全局状态（即使是重复）

所以，流水线寄存器阻塞会导致代码执行的总时钟周期数上升。

然而，指令的结果在写回寄存器之前，可能已经算好了。这样我们可以拿来用，减少阻塞的周期数。这就是 forward 转发。**转发不能对延迟有很大的影响。**规定转发的数据来源只能是流水线寄存器的输出。转发的 destination 可以是后面指令使用改操作数之前。

流水线冲突是本次作业最难的部分，需要详细的分析，特别是数据冲突。请关注以下几点：

- 第 i 条指令的结果被哪些指令使用时会产生冲突？请列举表格分析哪些冲突要使用阻塞、哪些冲突可使用转发。
- 可作为转发来源的流水段是 Memory （从 ALU 得出） 和 Writeback（从 ALU 得出，或从内存得出）。
- 可作为转发 destination 的流水段是 Decode （所有指令）和 Execute （使用 ALU 的指令）。
- 转发时，遇上阻塞该如何处理？比如，W 转发给 E，如果 M 阻塞了，W 写入寄存器，但 E 拿不到数据了。

本次实验可以不实现转发。

架构相关

本次实验要实现的指令为：addi xori ori andi lui jal beq ld sd add sub and or xor auipc jalr

汇编里可能现实的是伪指令，比如：

- load immediate(li) 指令，如果立即数较小，就相当于 x = r[0] + imm。（0号寄存器只读恒为0）

PC 寄存器的复位值为 `0x8000_0000`.

作业提交

DDL：**3月20日23:00** 与 **4月3日23:00**

提交内容：以学号命名的 `tar` 压缩包，如 `18307130024.tar`，包括 `vsrc` 和 `report.pdf`。

生成方法：

```
mkdir 18307130024
cp -r vsrc 18307130024
cp report.pdf 18307130024
tar cf 18307130024.tar 18307130024
```

第一次提交：你需要用五级流水线 CPU 在 `verilator` 仿真框架中跑通第一条指令。

第二次提交：仿真通过 `test1`，上板跑通测试 `test1`（上板现象：串口输出`Hello world!`，LED 灯亮）。

实验报告需要包括：

- 五级流水线的简易电路图，建议手绘拍照
- 分析流水线冲突的每一种情况，并给出解决方案，说明如何用电路实现

实验报告里不应有大段代码的复制。

通过测试+实验报告有上述内容，本次实验就可以满分。

实验报告里可以有：

- 对本门课程实验（文档、代码风格、视频录制等）的建议
- 对后续内容的期待

思考题

本次实验思考题不计分。

1. 前面讲到，关键路径会限制时钟频率，导致组合逻辑路径很小的部分也需要执行关键路径的时间。加法指令不需要用到 Memory 阶段的任何逻辑，但它也需要通过这一级流水。能否让它少走一级流水段？（好处：早一周期写入寄存器，缓解数据冲突问题）
2. bubble 是否需要对流水线寄存器的每一位都清零？这与电路布线相关，少接几根线可能就会改善关键路径的时延。