处理器设计进阶

杨倚天 黄嘉良

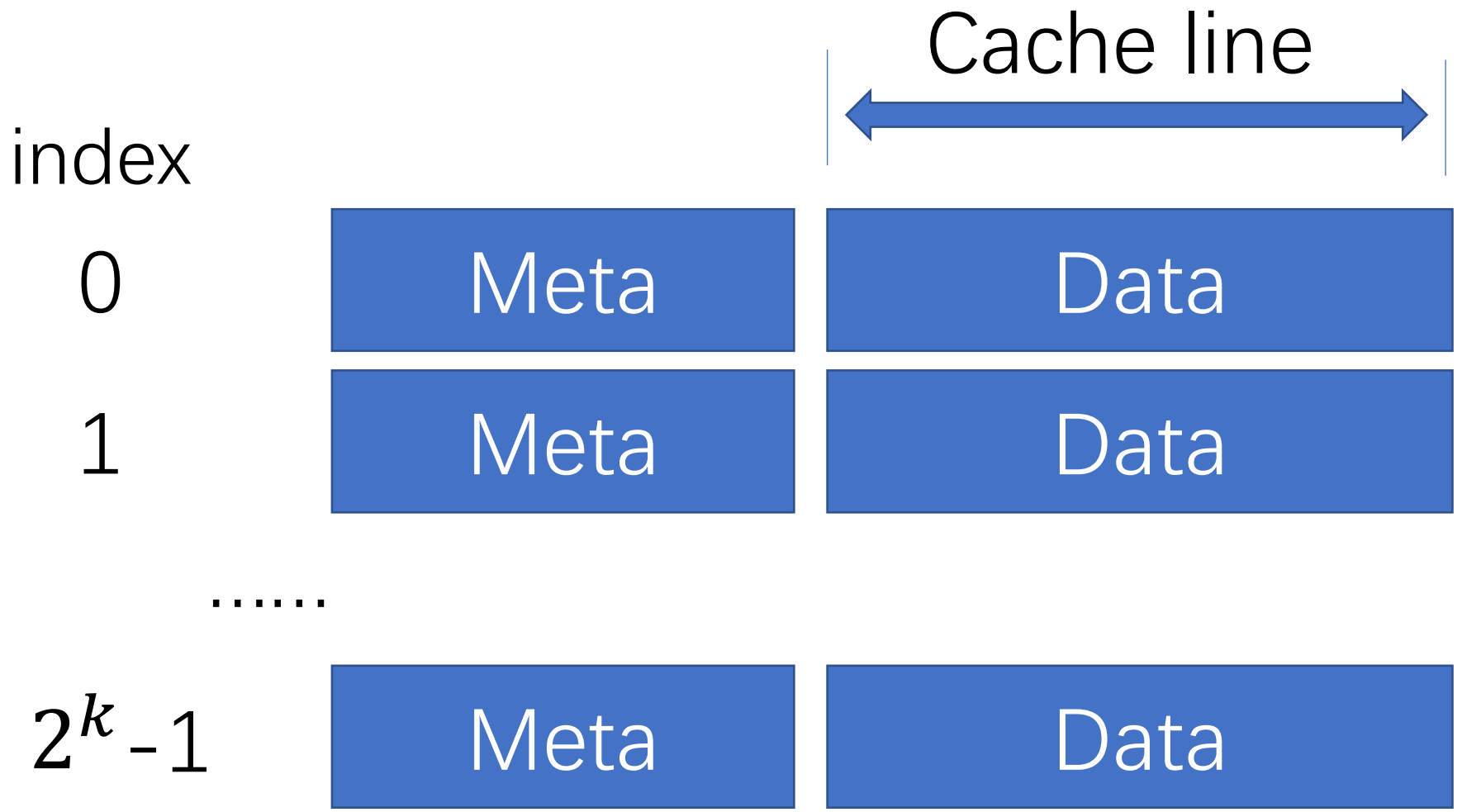
龙芯杯

- 欢迎有兴趣的同学参加第六届龙芯杯
- 不要怕太难了，只要有兴趣都不是问题
- 比赛正式时间是下学期的期末开始，基本持续整个暑假
  - 从下学期初期开始准备
- 可以直接联系任课老师

Cache

- 用更近更快的小ram来保存更远更慢的大ram中的一部分内容
- 设计上最关键的一点是，**当Cache命中的时候不能有气泡或暂停**
- 至于Cache不命中的开销没有必要纠结多等一个周期或稍等一个周期
- 数据Cache和指令Cache分别组织

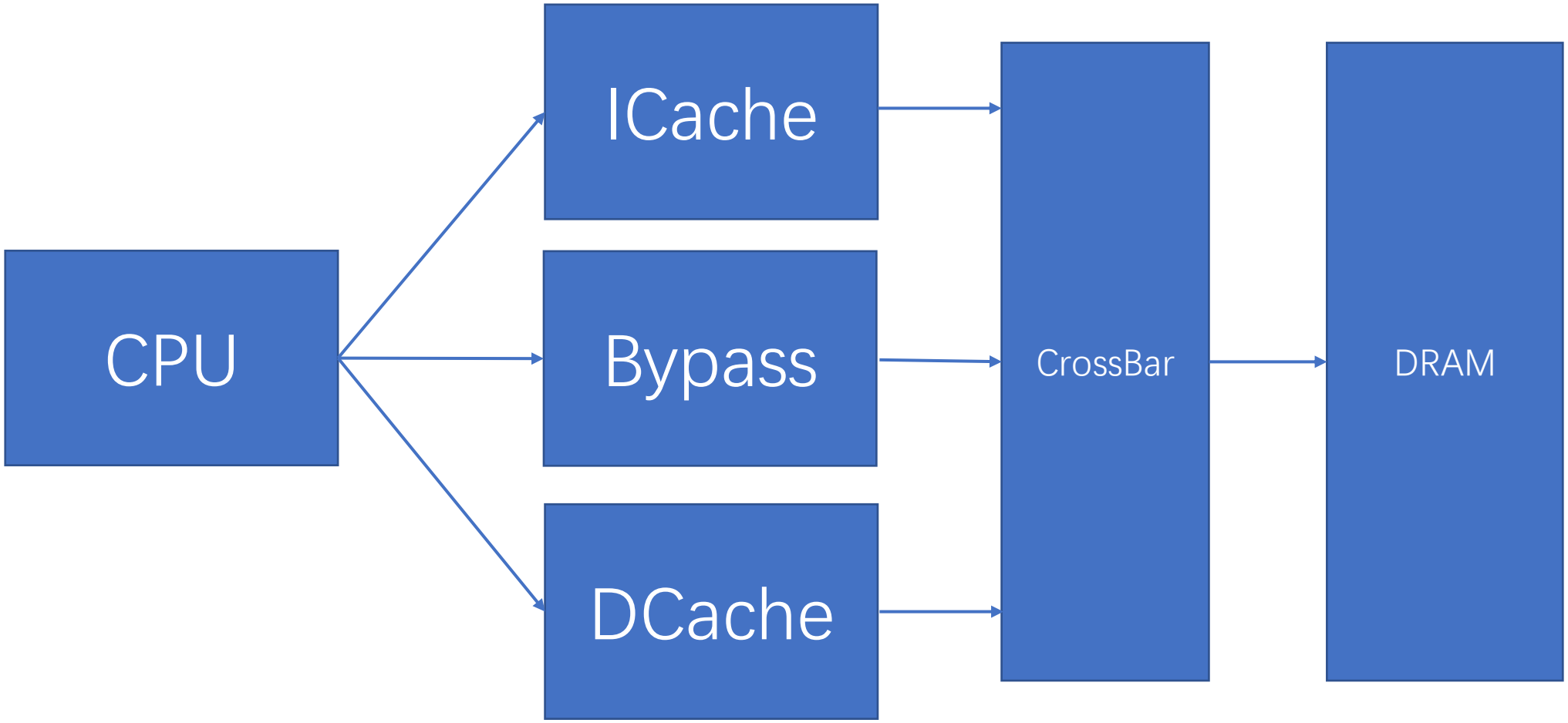
Cache



Cache

- 实现上先从直接映射开始，然后可以开始多路组相连
- 一个物理地址 = { tag, index, word_offset, 2'bxx }
- 通常采用VIPT(virtual index physical tag)，这样读cache ram和虚实地址转换可以同步进行
- 不过这样的策略也限制了单路Cache的最大容量，因为非虚实地址转换的部分就在一个page内
- 通常因为一个page的大小是4KiB，因此VIPT的Cache最大单路容量也就是4KiB

Cache



Cache

高性能（指cpu主频高）的Cache需要流水化设计，下面给出多路组相连一个参考（以读为例）

1. 第一个阶段cpu给index开始读
2. 第二阶段得到数据和Meta，比较tag，选出对应的Cache line，并根据是lb还是lw对读出的数据进行进一步修改

选数据有两个阶段的选：

- 选哪一路
- 选一个Cache line中的哪一个word

Cache

- 多路组的替换算法：闻名遐迩的LRU，实现上通常是Pseudo-LRU
- 硬件上实现替换算法可以采用**树形的结构**来组织
- 对于二路组相连，Pseudo-LRU和LRU应该是一样的

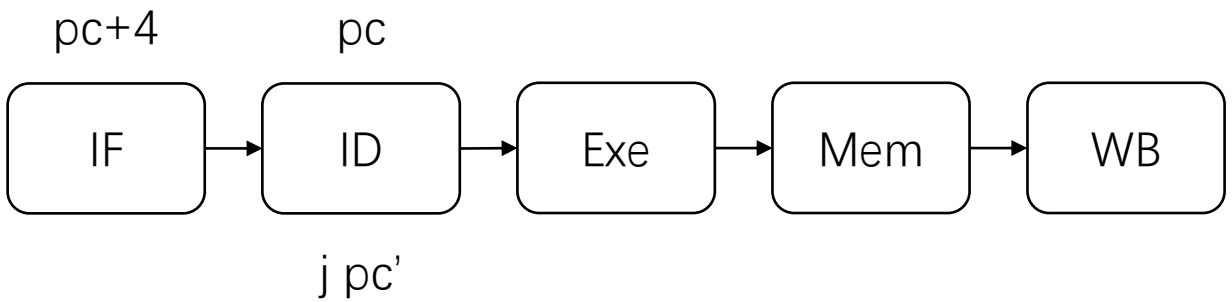
Cache

- DCache和ICache很不相同，因为ICache只读，**Dcache支持写**
- 写策略有：write-through和write-back，感觉是write-back效率更高？
- 对于**写不命中：** write allocate和No-write allocate，感觉write allocate效率更高，即先读进来一个Cache Line再修改对应的数据后存到Cache里

Cache优化

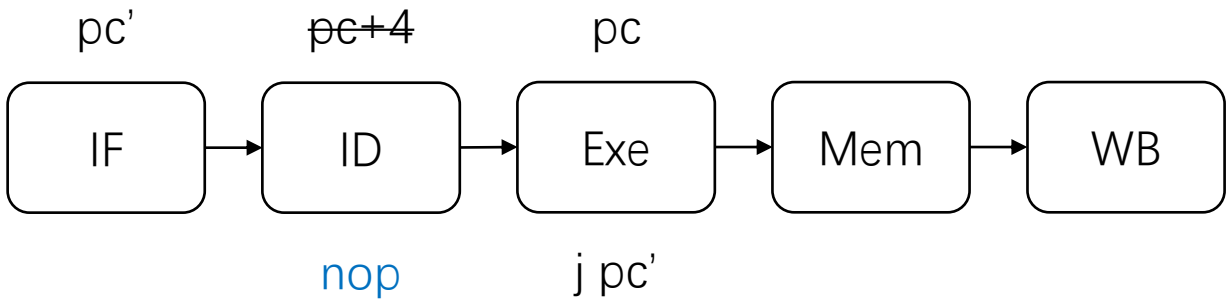
- DCache一个优化是Write Buffer，把要写回的数据放到Write Buffer中，只要write buffer没满就不会阻碍后续流水线的运行
- 还有Victim Buffer，对于替换出来的缓存行，可以不马上扔掉而是放到一个buffer中，后续可能很快又命中这个Victim Buffer
- Xilinx的芯片内部有延迟很低的Block RAM（应该有几MiB的大小），当然也有用查找表搭出来的LUT RAM，它们都可以做到一个周期读出要求地址内的数据。

分支预测



朴素实现中，
1. 要等到 ID 阶段才知道是否是跳转指令，
2. 因此要得知是否跳转得等到 ID 阶段结束。

此时 IF 阶段已经在取指令，
拿错指令的可能性无法避免。

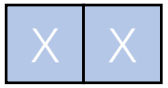


分支预测需要做到：
1. 尽早判断出是否为跳转指令，（利用 pc）
2. 预测出是否跳转并给出地址，（多种预测策略）

才能避免产生气泡。

分支预测

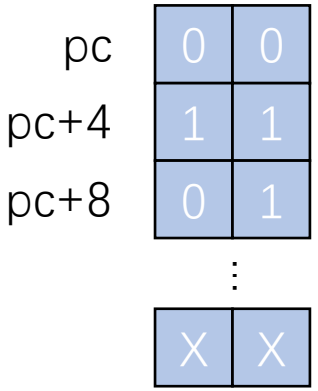
两位分支预测器可以处理部分浅层循环嵌套的情况。



为什么无法处理深层循环嵌套?
原因在于它是全局分支预测器,
对外层循环的预测被里层循环干扰了。

```
for(int i = 0; i < n; i++) {
    for(int j = 0; j < m; j++) {
        for(int k = 0; k < t; k++) {...}
    }
}
```

如何处理?
对每条指令单独进行预测即可。



如何压缩空间?
利用局部性原理，做成直接相连缓存类似的结构。

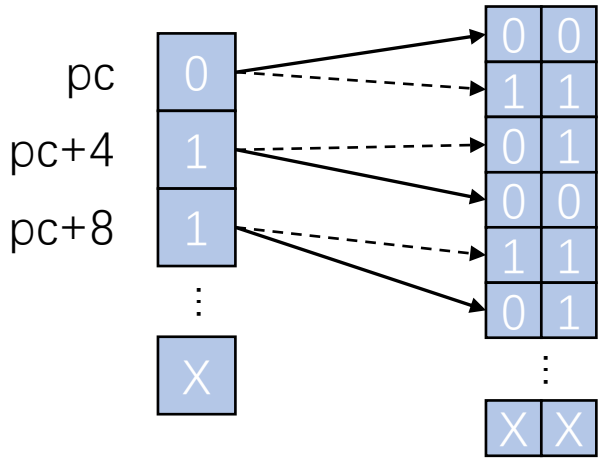
分支预测

如果是这种代码会怎样?

```
for(int i = 0; i < n; i++) {
    if(i & 1) {...}
    else {...}
}
```

局部分支预测仍然有缺陷，准确率只有 50%。

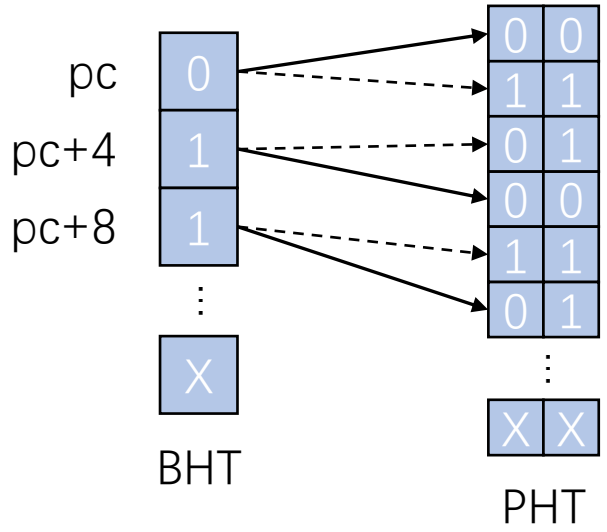
找规律，一次跳转，一次不跳，一次跳转，一次不跳…
如果能够记录下上一次是否跳转，就能够预测下一次是不是要跳转了。历史跳转信息能够提供帮助。



用 1bit 记录下上次有没有跳转，
再根据跳转没跳转来选择分支预测器。

如果需要更强的找规律能力，
则可以记录更多 bit 的历史跳转信息。

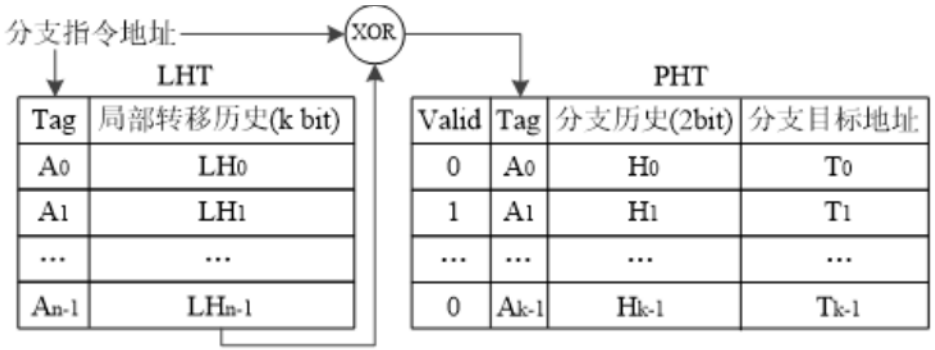
分支预测



当记录的 pc 数量增大、记录的历史 bit 数量增加时，
PHT 的大小将会飞速增加。

但是一般人写的代码中不会有特别复杂的跳转规律，
也就是说历史跳转的循环节不会很长，
也就是说实际上 PHT 中很多条目都用不到。

因此考虑复用 PHT。
因为我们要找的 PHT 条目是由 pc 和 BHT 条目唯一确定的，
而对这种离散的映射关系有一种很好的压缩空间的办法，
那就是散列。



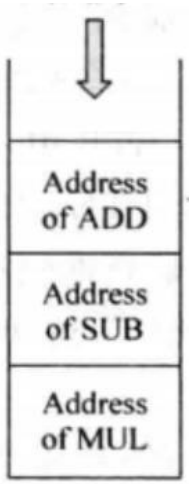
将 pc 和查询到的 BHT 条目进行 xor 操作，
再使用这个值去查询 PHT 条目，
是一种比较简单的做法。

分支预测

以上的分支预测实现有个前提，就是分支地址固定。
如果分支地址不固定，要怎么做？

什么情况下分支地址不固定？j ra
这是一条用于函数调用返回的指令。
1. 函数调用的返回地址是调用函数的地址+4。
2. 函数调用是栈式的。

使用栈来记录所有的调用函数的地址+4，
遇到 j ra 时，就弹栈。



RAS

双发射流水线

一个核的 CPU 不够用时，那就再给它来个核。
流水线的主频到极限了，那就再给他来条流水线。

一条流水线之间的依赖关系已经很麻烦了，两条流水线那不逆天？
因此首先要把封装做好，分开考虑流水线的内部处理和外部依赖。

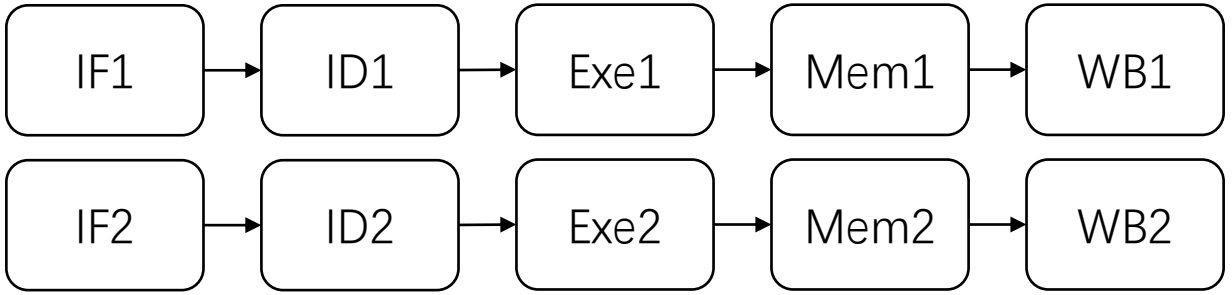
也不难。
除了麻烦以外，和一条流水线是类似的。

对于每个流水段来说，他需要一些接口：
can_output // 能否输出自己阶段的处理结果
assign_flush // 是否需要作废自己阶段的处理结果
has_input // 是否有数据等待输入
…

这么一来，处理外部依赖时，只需要将对应的控制信号赋值就行，而不需要考虑内部逻辑，可以简化思考的复杂度。

数据来不及前传或工作没做完？can_output = false
需要清空流水线？assign_flush = true
…

双发射流水线



双发射流水线实现上的一些考虑:

1. 要注意指令执行的顺序。

　　可以强制 IF1 的 pc < IF2 的 pc，这样就只需要让第 2 条流水线的各流水段不先于第 1 条流水线对应的流水段即可。

2. 数据前传需要跨流水线。

3. 容易发生结构冲突。

　　像 IF、Mem 这两段流水线，都需要同时访问内存。IF 的处理简单，因为指令流一般连续，可以同时取出两条指令。但是 Mem 就需要注意只能允许一条指令通过，由于在 1 中限制了 pc 的关系，所以这里可以优先 Mem1，然后 Mem2，设置好 can_output 就没问题。

　　又像 WB 这段流水线，同时需要访问寄存器组，就需要注意写同一个寄存器时的先后。

Cache和多发射

- CPU性能的优化的重点还是在于**分支**和**访存**指令
- 多发射至少你能够一个周期拿到两条指令，Cache的作用就比较重要：每次读一个line就可以返回两条指令
- 可以在取指和译码中间加一个指令队列（尚且不太清楚指令队列的优化效果）
- Cache中的RAM最多只有两个端口（也就是只能同时读出两个数据），对于**多发射**而言如何多个访存同时执行?
- 一个设计思路是把一个Cache Line再拆开成多bank组织起来，只要两个读写请求不在一个bank上就可以同时进行（但是tag可能需要完全复制一份）

资料推荐

- 《超标量处理器设计》清华大学出版社，姚永斌
  - 学校图书馆有实体书，图书馆网站上可以在线阅读
- Xilinx文档中关于RAM Macro的介绍
  - https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug953-vivado-7series-libraries.pdf
  - 重点查看第二节Xilinx Parameterized Macros一节关于RAM的说明