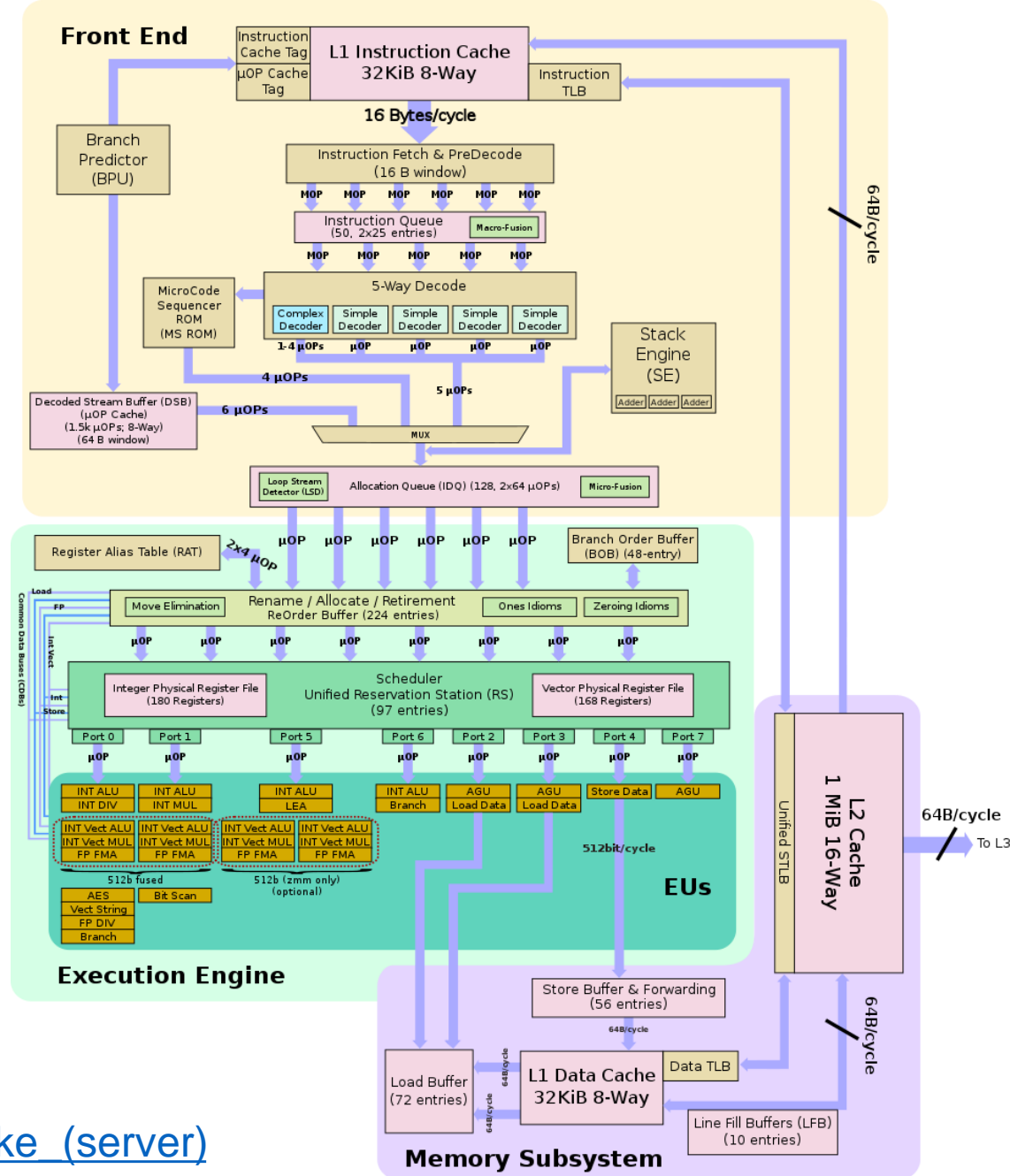
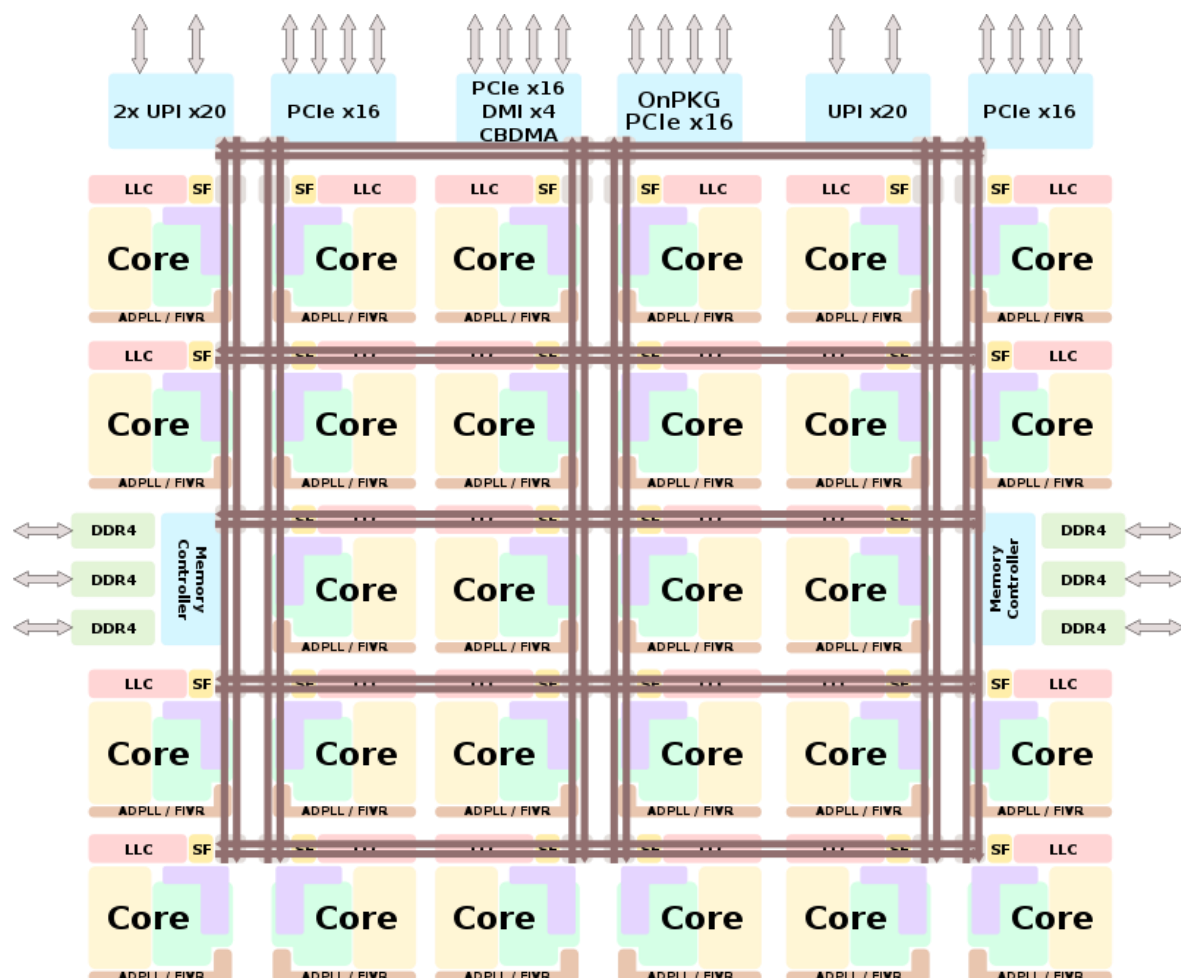


# 处理器优化

郭健美

2024年秋

# Intel Skylake Microarchitecture

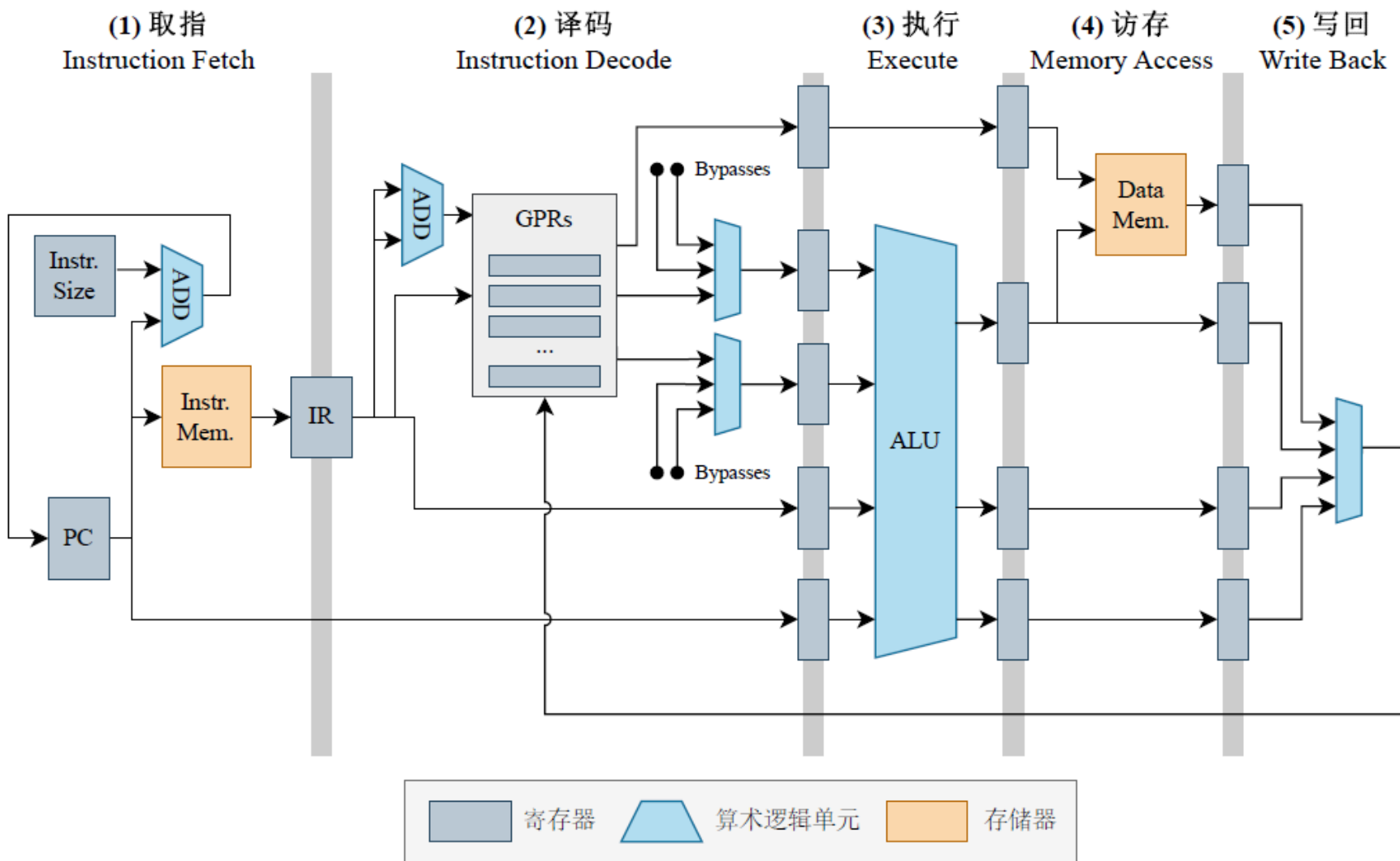


[https://en.wikichip.org/wiki/intel/microarchitectures/skylake\\_\(server\)](https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server))

# 内容

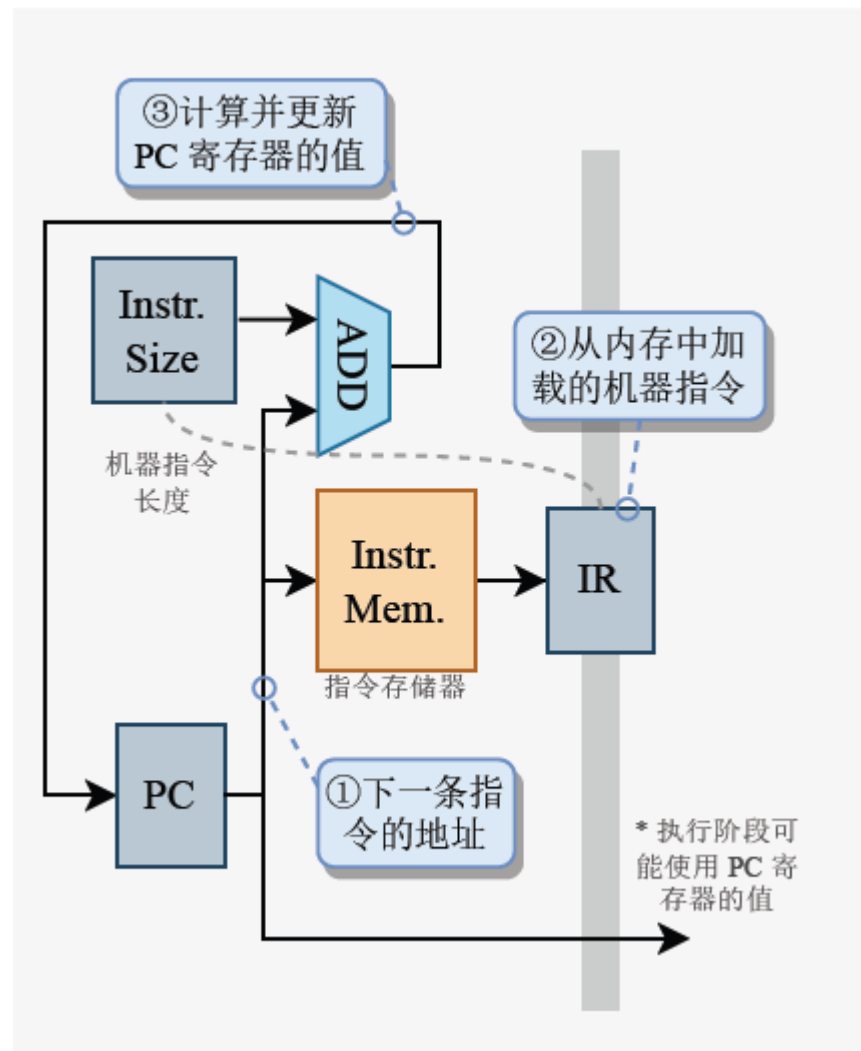
- 五阶段处理器
- 流水线执行
- 超标量处理
- 乱序执行
- 推测执行

# 五阶段处理器



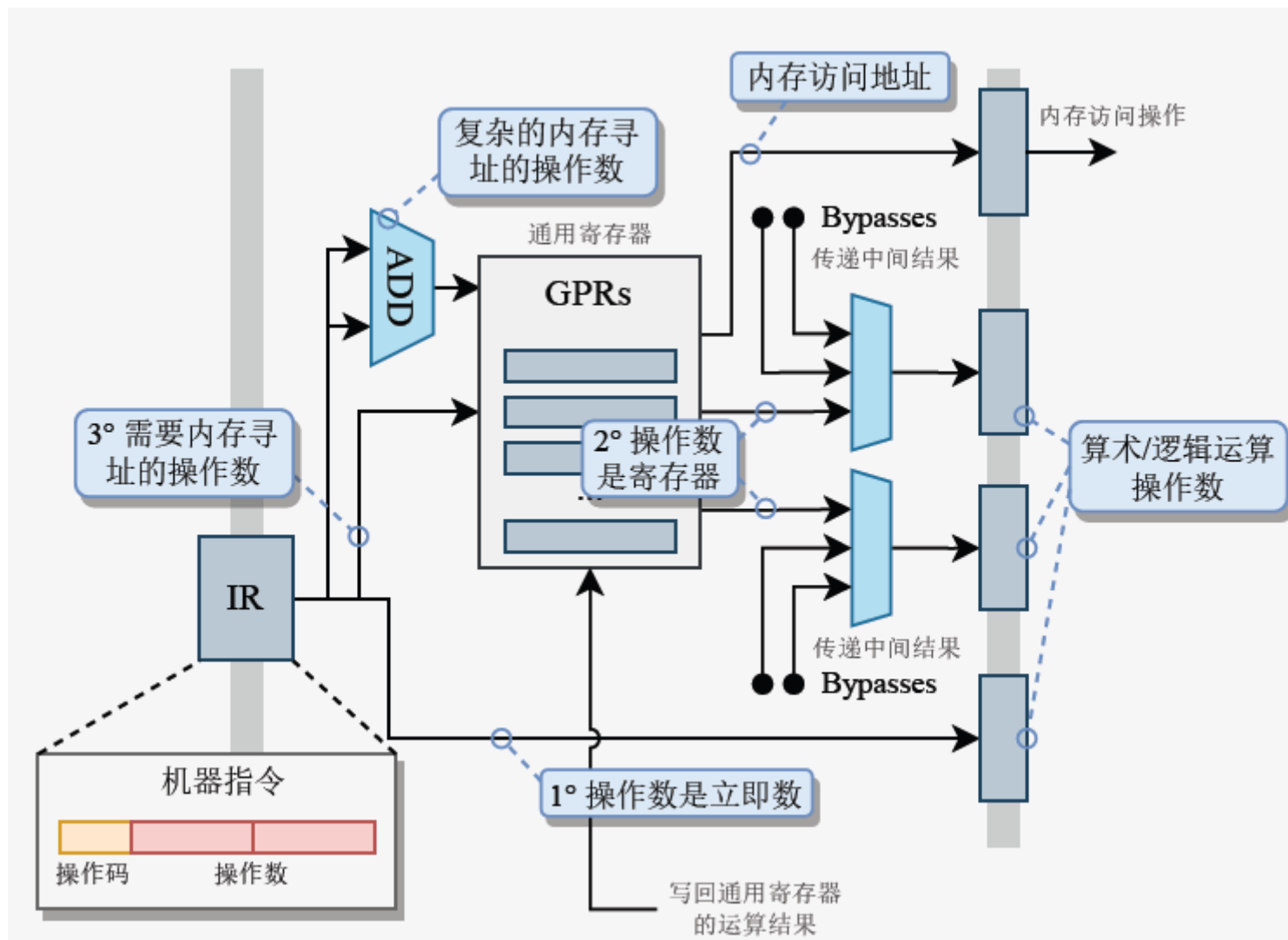
# 五阶段处理器：取指

- 程序计数器 (Program Counter, PC) 是存放将要被执行的下一条指令地址的寄存器，取指单元 (fetch unit) 根据程序计数器指向的地址，从存储器中读取指令并加载到指令寄存器 (Instruction Register, IR)，之后程序计数器自增



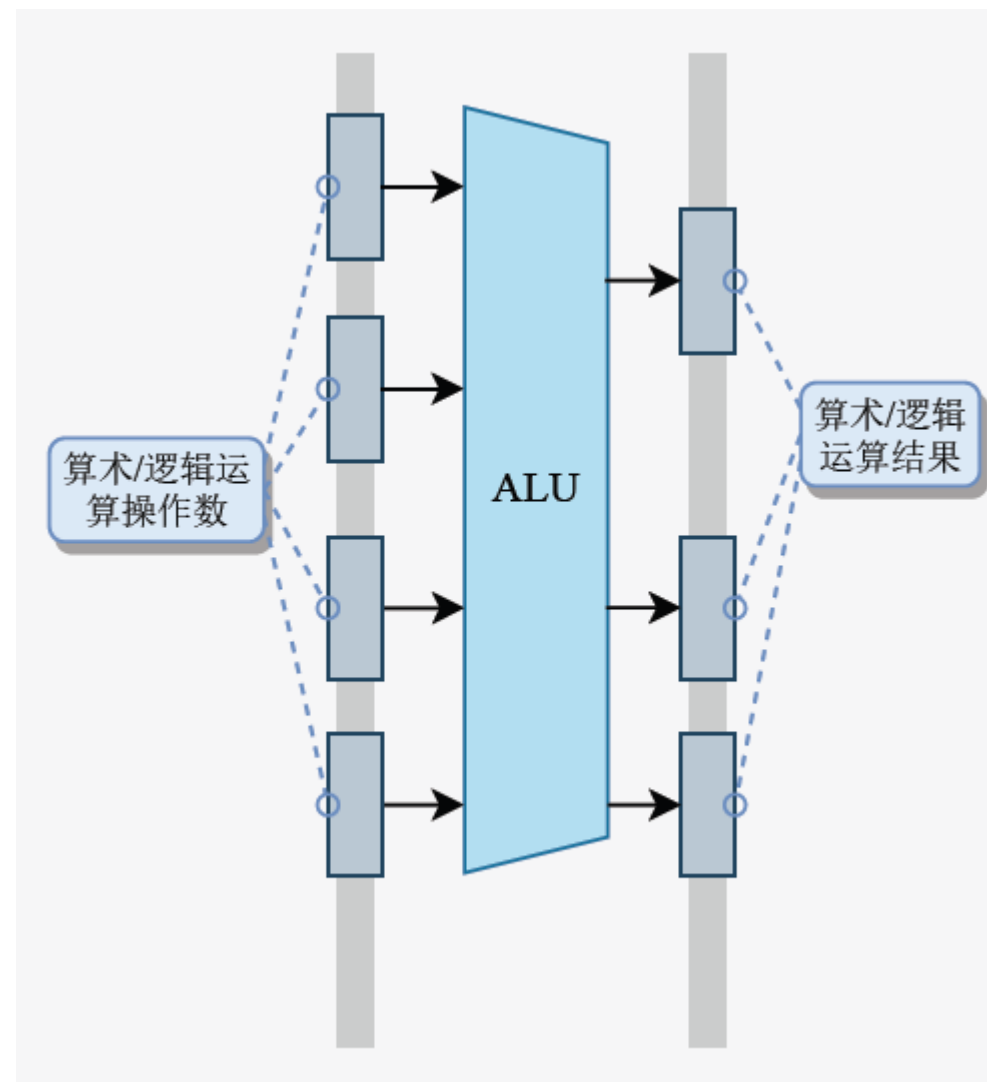
# 五阶段处理器：译码

- 译码器 (decoder) 读取指令寄存器中的机器指令，分析指令中涉及的操作数 (operand) 及其寻址方式，准备执行阶段所需要的数据



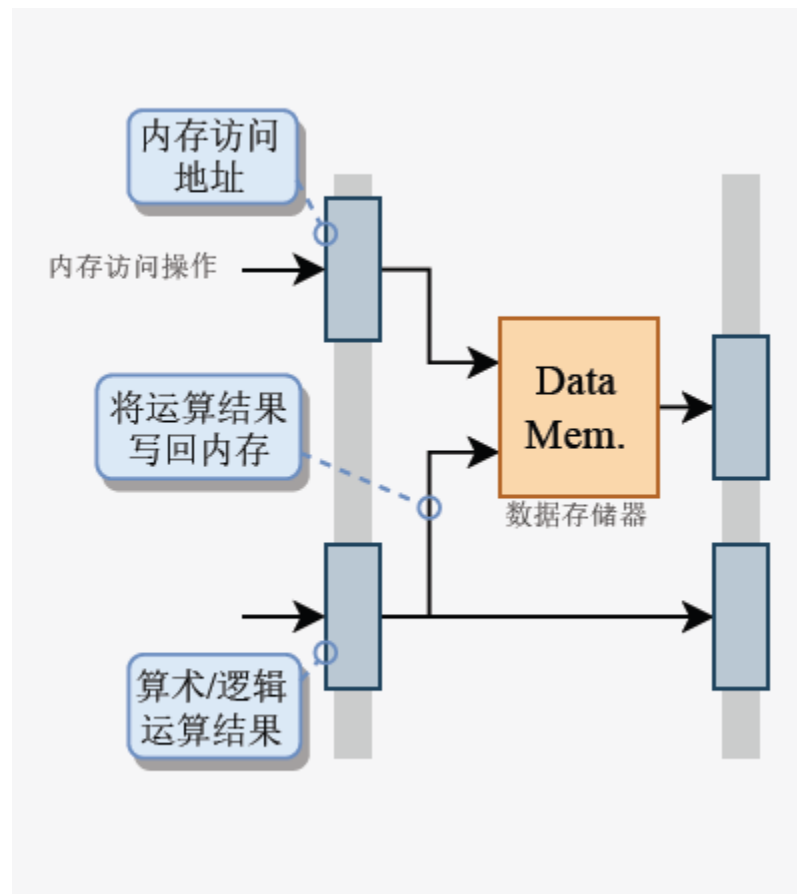
# 五阶段处理器：执行

- 译码阶段的操作数准备好了之后，算术逻辑单元（Arithmetic Logic Unit, ALU）根据机器指令的操作码（operation code, opcode）执行对应的算术逻辑操作，输出操作结果



# 五阶段处理器：访存

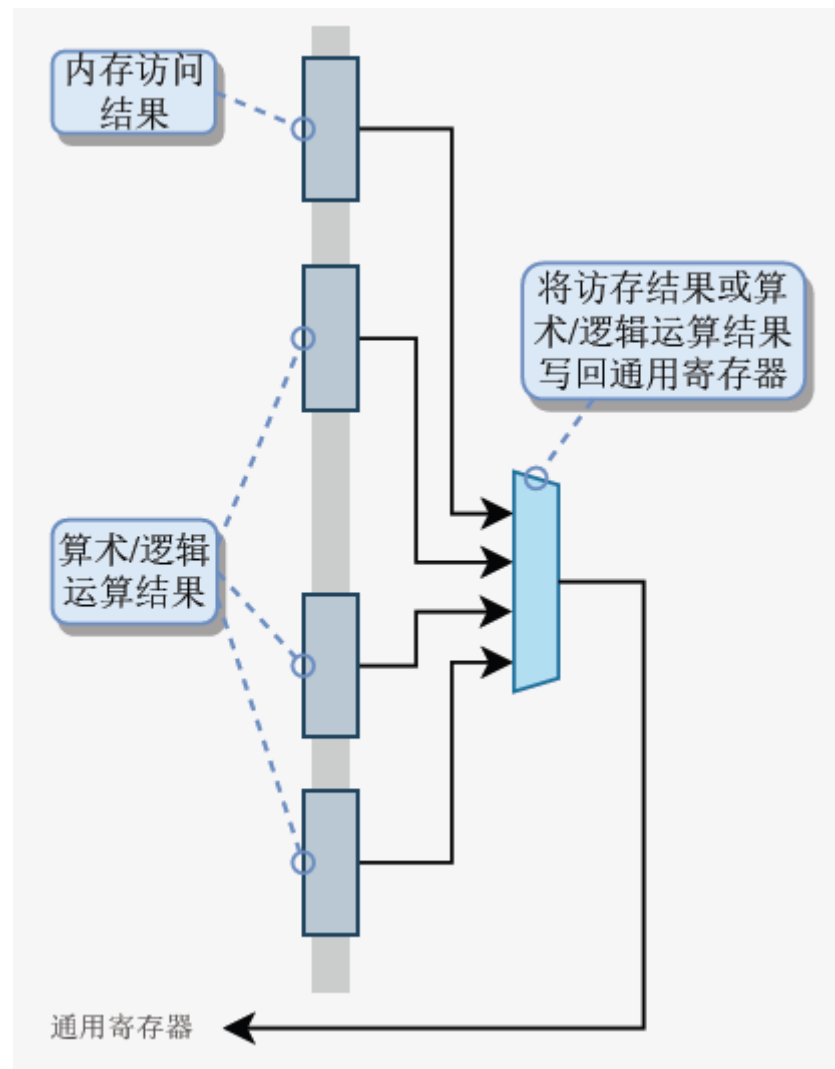
- 主要负责从存储器加载 (load) 数据或存储 (store) 数据到存储器



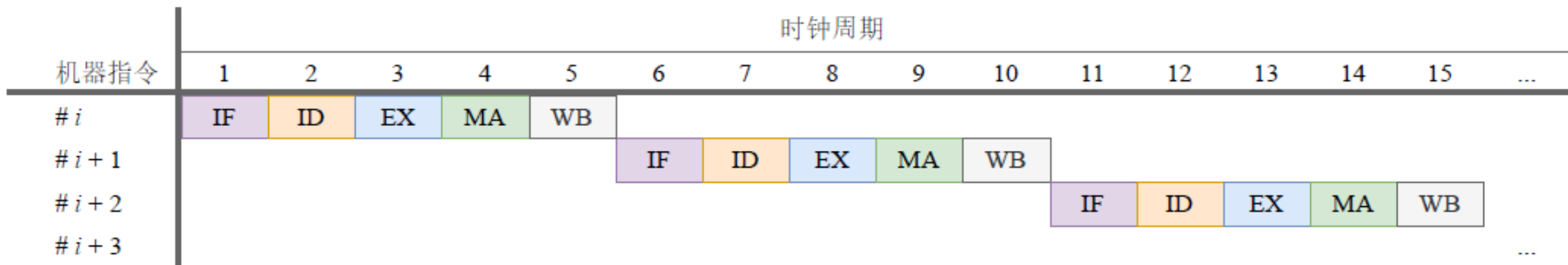


# 五阶段处理器：写回

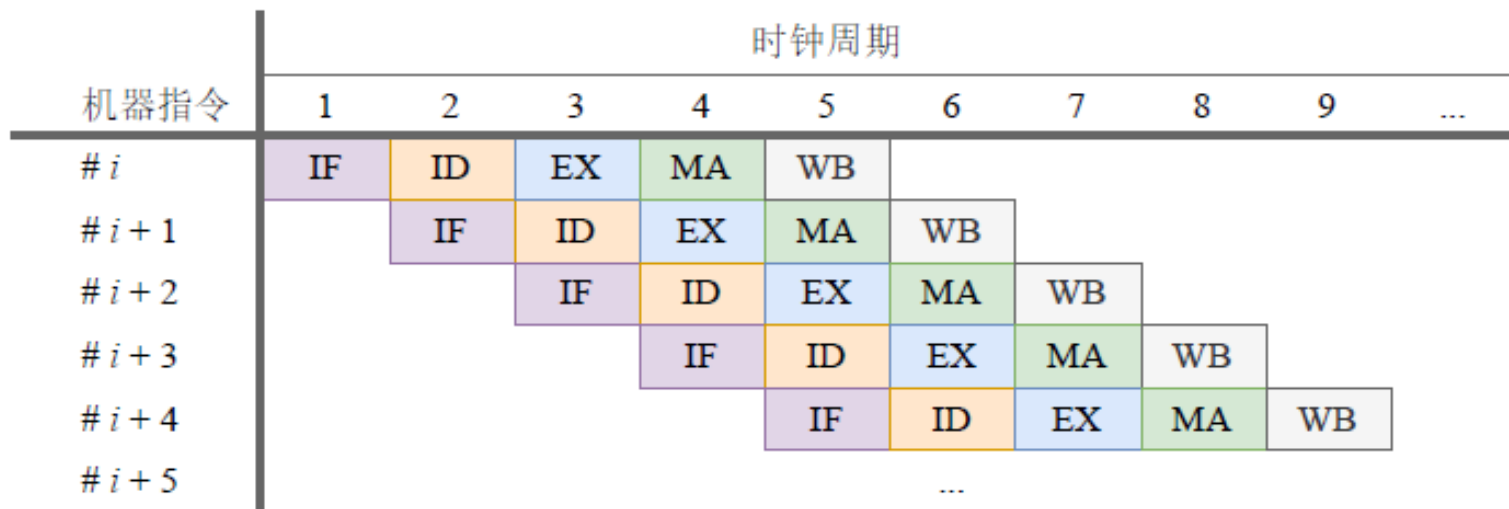
- 将访存阶段加载的操作数或将执行阶段的运算结果写回通用寄存器，并更新系统状态



# 流水线执行：指令流水线



(a) 顺序执行



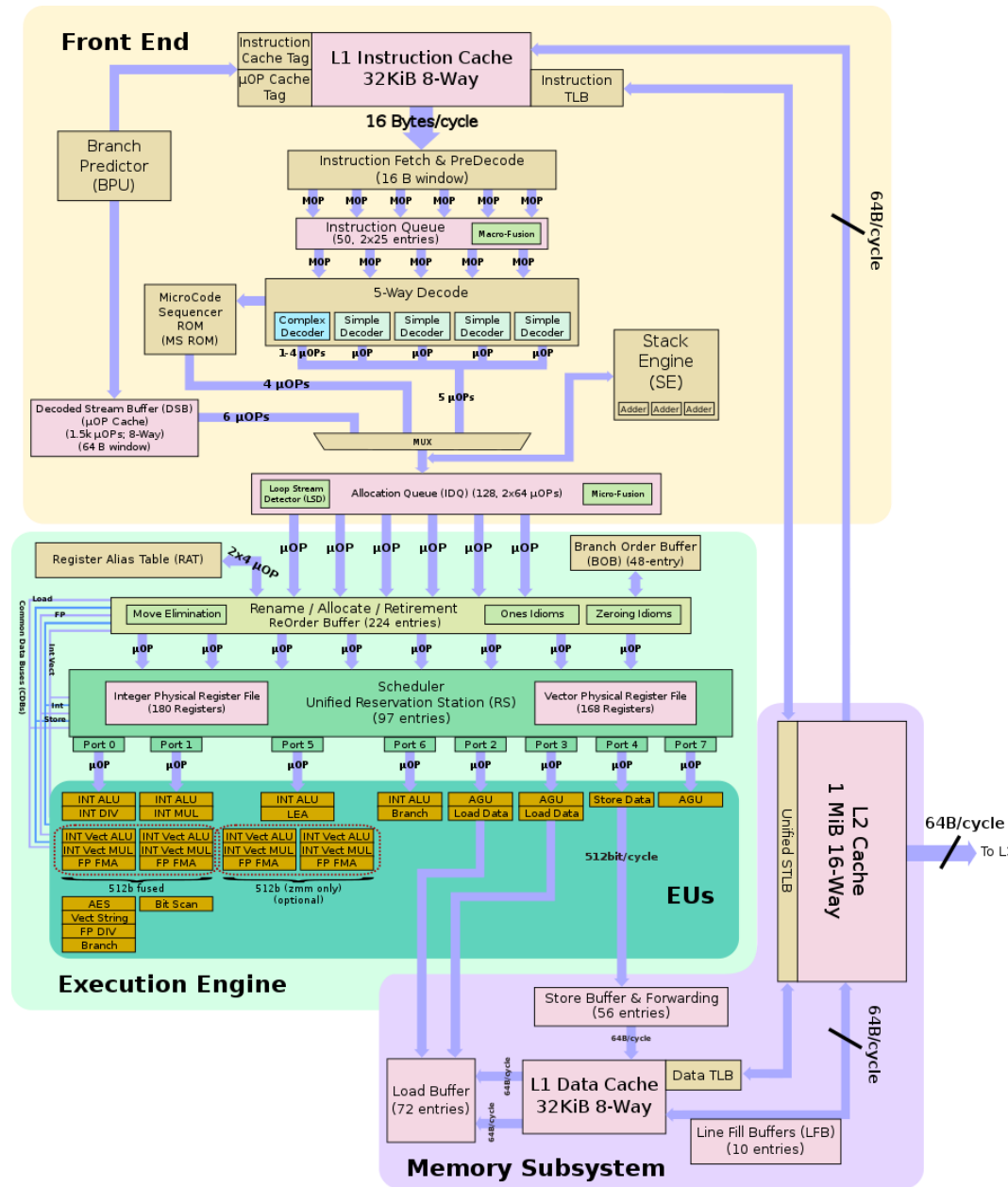
(b) 流水线执行

# 流水线执行：指令流水线

- 将机器指令在处理器内部的执行过程划分为不同的阶段，其本质目的是为了将执行过程流水线化，形成指令流水线
- 在指令流水线中，多条指令的执行阶段在时间上重叠，处理器每一个阶段的组件能够并行处理不同指令
- 流水线使得处理器各个阶段组件能够得到有效利用，使得处理器执行机器指令的吞吐量有显著提高

# 流水线执行：前端与后端

- 前端的主要任务是从内存中加载机器指令并解码，为后端的执行做准备
- 后端的主要任务是负责机器指令的实际执行，包括对数据进行运算、访问内存以及将结果写回寄存

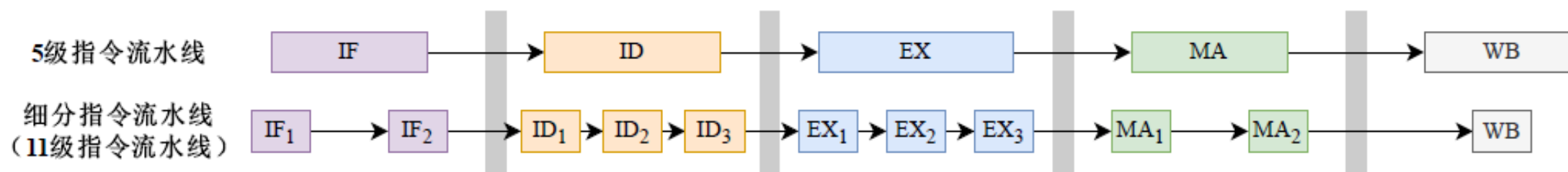


# 流水线执行：性能评价

- 每指令的平均时钟周期数 (Cycles Per Instruction, CPI)
- 每时钟周期的平均指令数 (Instructions Per Cycle, IPC)
- 顺序执行 vs 流水线执行
  - 对于顺序执行，每 5 个时钟周期完成一条指令的执行，此时  $IPC = 1/5$
  - 对于流水线执行，当流水线满载时，每 1 个时钟周期完成一条指令的执行，此时  $IPC = 1$ ，相较于顺序执行，指令执行效率提升了 5 倍
- 然而，实际中处理器各个阶段的执行很难在 1 个时钟周期内完成

# 流水线执行：细分

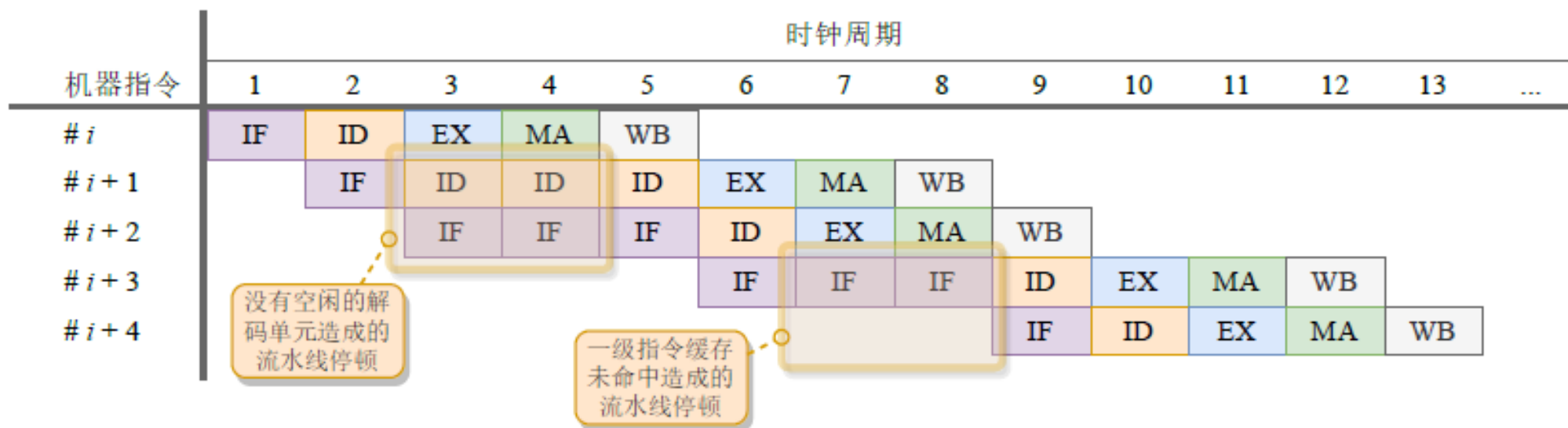
- 简化各个阶段的任务，进一步细分阶段



- 提高指令级并行度：更多指令能够同时处于不同阶段，允许同一时刻处理多个指令的不同阶段，从而提高处理器执行指令的并行度
- 有助于提高时钟频率：细分后，各个阶段的任务更小，实现的电路更简单，有助于处理器以更高的时钟频率运行

# 流水线执行：停顿

- 理想情况下，每一条指令在每一个时钟周期结束时都能够进入下一个阶段，此时能够达到最大吞吐量。但实际上，指令在流水线中可能出现停顿（stall）。



# 流水线执行：冒险

- 流水线停顿实际上是规避流水线冒险 (hazard) 的一种手段
- 流水线冒险泛指指令流水线中部分指令之间存在依赖时可能引发的问题，这些问题可能会导致指令无法在预定的时钟周期内执行。
  - **结构冒险 (structural hazard)**：因缺乏硬件资源而导致指令无法在预定时钟周期内执行，例如，指令流水线的某个功能单元在同一时刻被多个指令需要。
  - **数据冒险 (data hazard)**：因无法提供指令所需数据而导致指令无法在预定时钟周期内执行，例如，一个指令依赖于另一条指令的结果，而这个结果尚未计算完成，需要等待计算结果完成，否则会导致计算错误。
  - **控制冒险 (control hazard)**：也称分支冒险，主要由条件分支 (conditional branch) 指令引起。当改变控制流的条件尚未确定时，流水线不明确应当选择执行哪一条分支的指令流，因此取到的指令可能并不是所需要的，此时将导致正确的指令无法在预定的时钟周期内执行。



# 流水线执行的三大冒险与优化



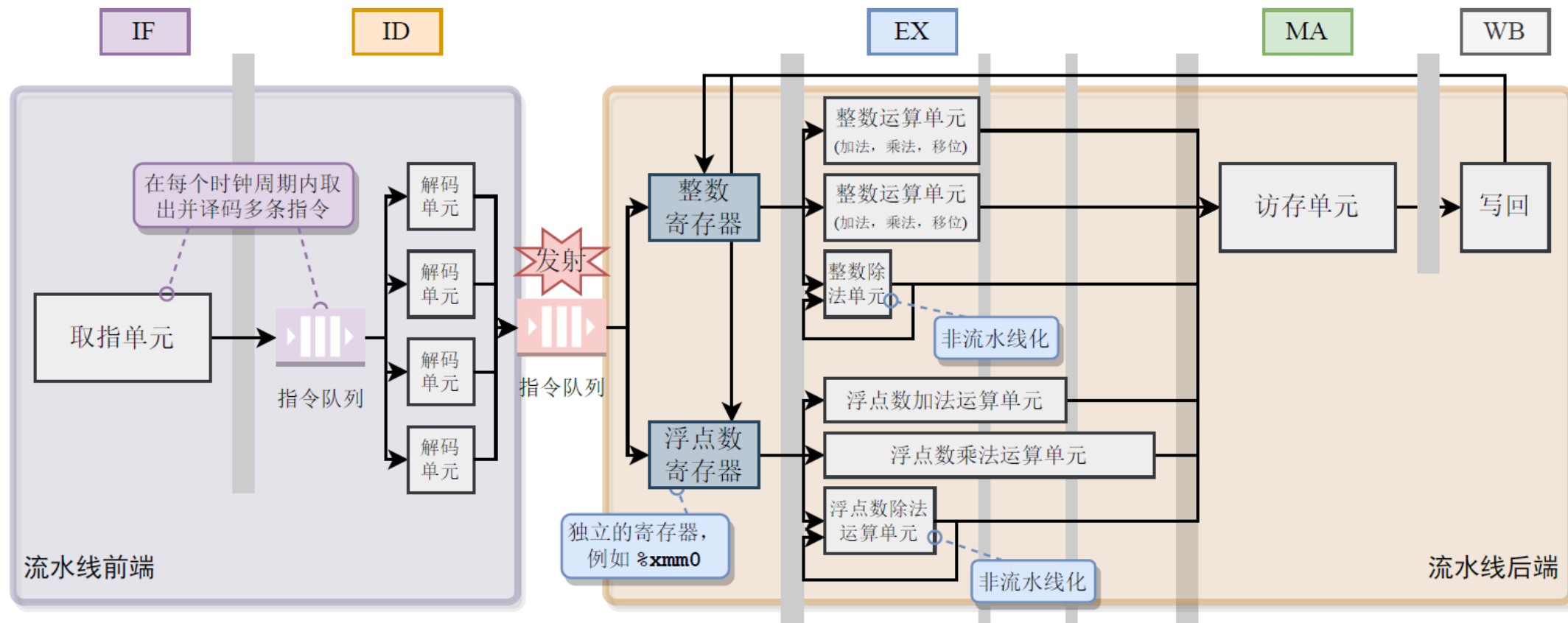
# 超标量处理

- 1. 结构冒险：因缺乏硬件资源而导致指令无法在预定时钟周期内执行，例如，指令流水线的某个功能单元在同一时刻被多个指令需要。
- 2. 不同类型指令的复杂程度不同，在执行阶段消耗的时钟周期数也不同。

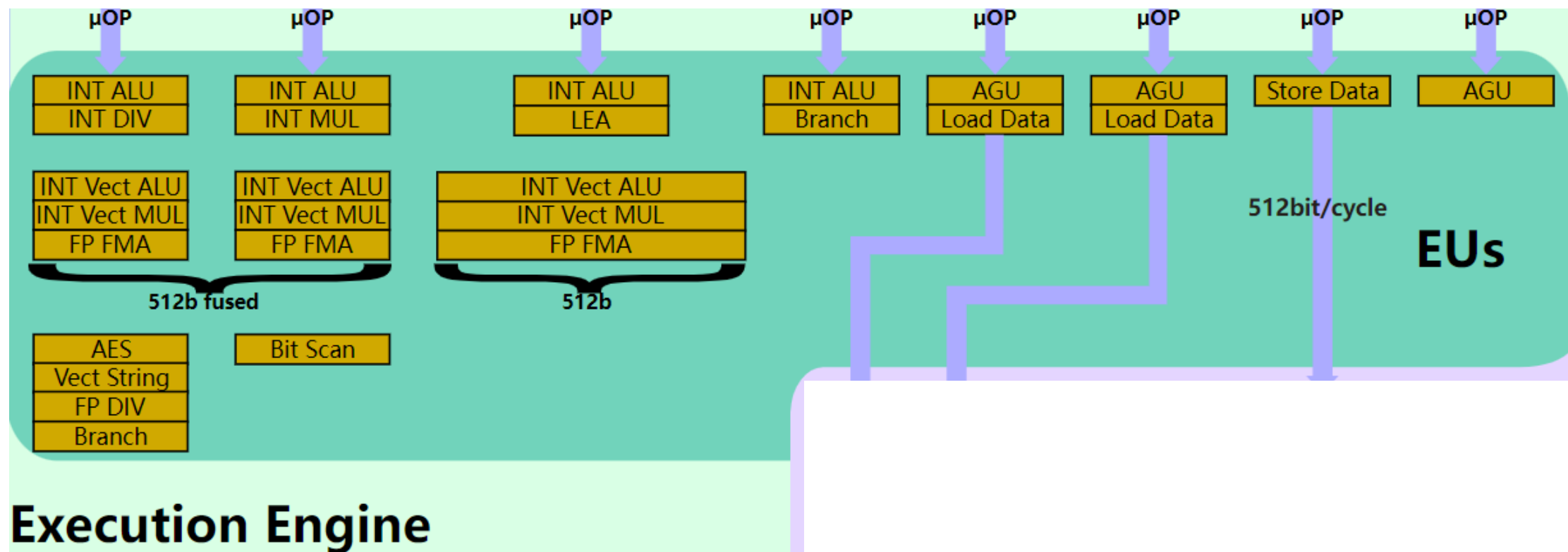
指令类型	x86-64 指令示例	执行阶段的延迟 (以时钟周期计)
整数算术运算、逻辑运算、移位指令	add, sub, and, or, xor, sar, sal, lea ...	1
整数乘法指令	mul, imul	3
整数除法指令	div, idiv	可变
浮点数加法指令	addss, addsd	3
浮点数乘法指令	mulss, mulsd	5
浮点数除法指令	divss, divsd	可变
浮点数乘积累加（Fused Multiply-Add, FMA）运算指令	vfmass, vfmasd	5

# 超标量处理

- 多个执行单元使得同时执行多条（同类型/不同类型）指令成为了可能，进一步提高了指令级并行性



# 超标量处理



[https://en.wikichip.org/wiki/intel/microarchitectures/skylake\\_\(server\)](https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server))

# 机器指令与微操作

- 为了更高效地实现超标量流水线，便于发射单元的处理以及后端指令级并行的实现，对于x86-64 架构的处理器，会在译码阶段将机器指令拆解为更加简单的操作，称为**微操作**（micro-operation, micro-op 或  $\mu\text{op}$ ）
- 例如，Intel Haswell 处理器的前端能够在每个时钟周期内向后端发射4 个微操作

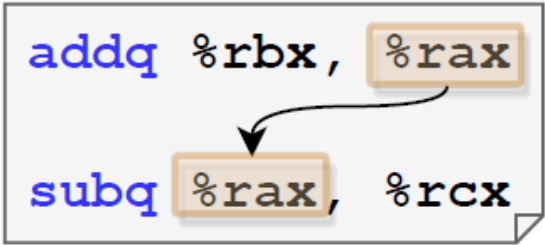
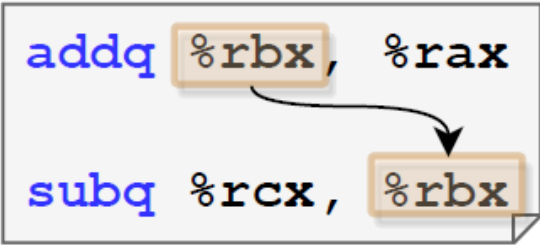
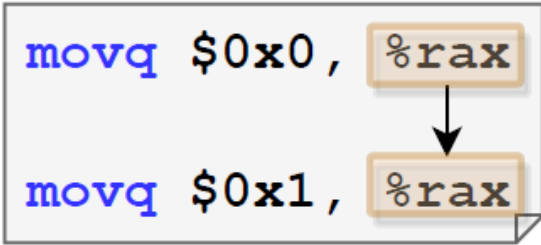
# CISC vs. RISC

- 随着技术的发展和经验的积累，两者之间的明确边界逐渐模糊，而结合两者思想的架构设计逐渐普遍

	CISC	RISC
指令集复杂度	复杂，包含许多复杂而功能强大的指令，一条指令可以执行多个低级操作	精简，通常每条指令只执行一个基本操作
指令执行时间	部分指令需要消耗较多时钟周期来执行，例如将一个整块从内存的一个地址复制到另一个地址的指令，以及同时操控多个寄存器的指令	追求简单性和效率，每条指令通常在一个时钟周期内执行完毕
指令格式	指令编码长度可变，寻址方式多样，包括偏移量、基址、变址、伸缩因子等	指令编码长度固定，寻址方式简单，通常仅支持基址与偏移量寻址
内存访问	可以直接对内存的操作数进行操作，因此指令可能包含对内存的直接访问，甚至可以执行复杂的内存操作	仅通过 Load / Store 型指令进行内存访问，要求将数据加载到寄存器中进行处理

# 乱序执行

- 数据冒险：因无法提供指令所需数据而导致指令无法在预定时钟周期内执行，例如，一个指令依赖于另一条指令的结果，而这个结果尚未计算完成，需要等待计算结果完成，否则会导致计算错误。
- 数据依赖（data dependence）的类型

真依赖（true dependence）	假依赖（false dependence）	
写后读（Read After Write, RAW）  前一指令写入数据后，后一指令读取该数据	读后写（Write After Read, WAR） 反依赖（anti-dependence）  前一指令读取数据后，后一指令在同位置写入新的数据	写后写（Write After Write, WAW） 输出依赖（output-dependence）  前一指令写入数据后，后一指令在同位置写入新的数据

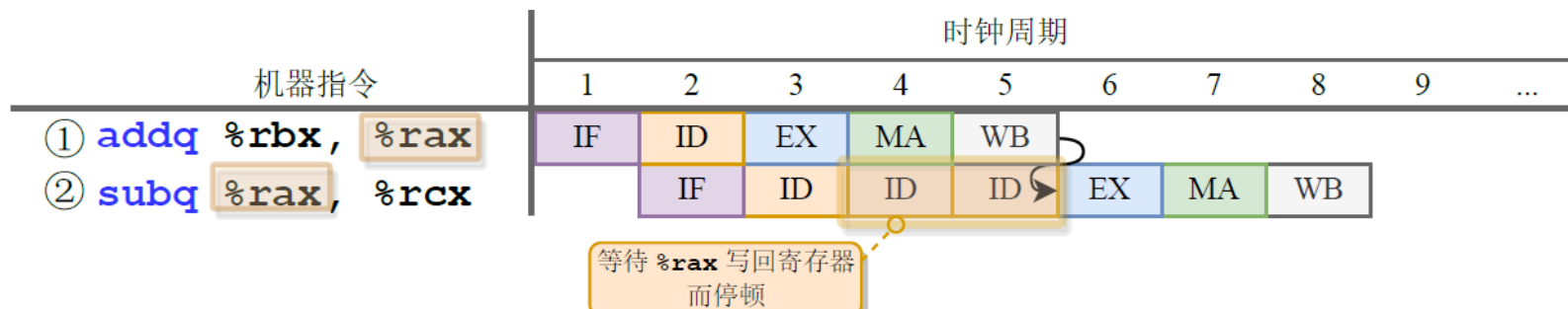
# 乱序执行

- 现代处理器主要借助乱序执行（Out-of-Order Execution, OOE 或 OoOE）等技术改造指令流水线，以缓解数据冒险造成的流水线停顿
- 对于真依赖，能够通过**旁路**（bypassing）和**乱序执行**的技术以减少其造成的性能损失，但是没有办法完全地避免真依赖造成的流水线停顿
- 对于假依赖，能够通过**寄存器重命名**（register renaming）机制完全消除假依赖造成的流水线停顿

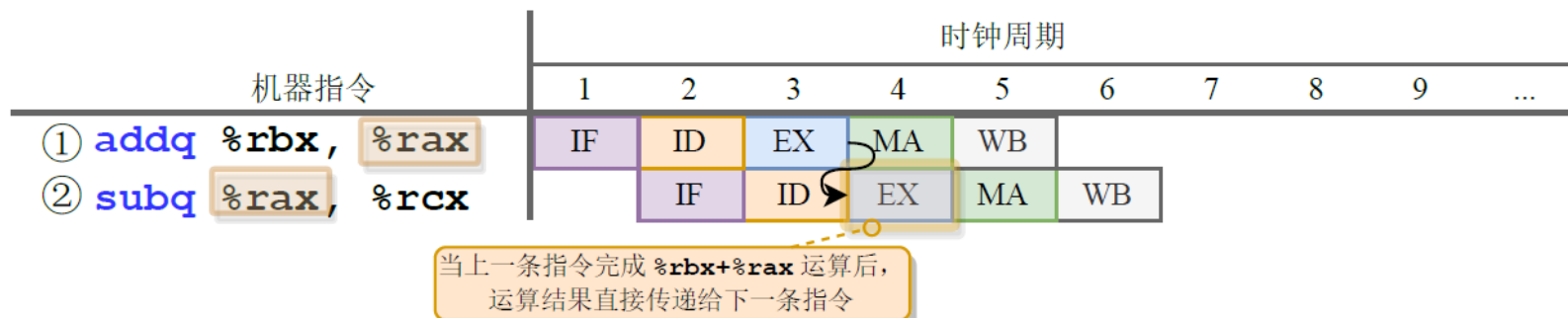


# 旁路 / 前递

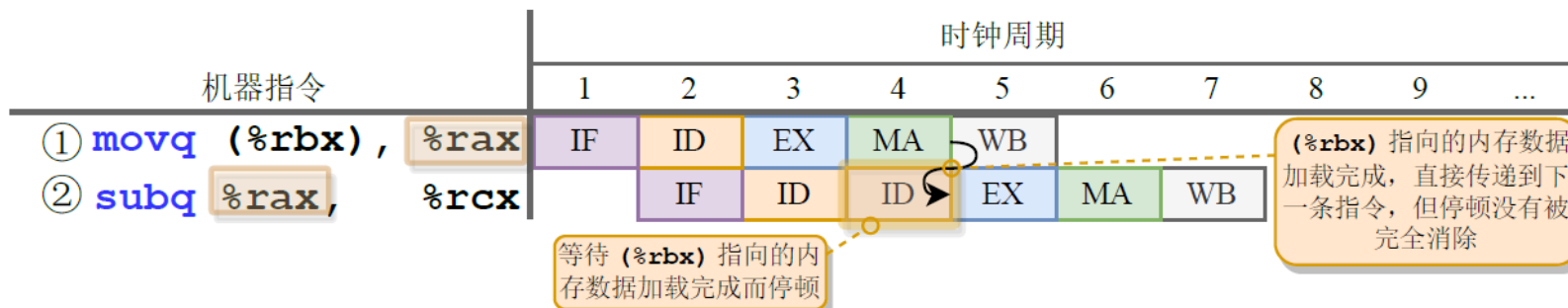
- 直接从前一条指令的中间阶段将数据传递给后一条指令



(a) 因 RAW 依赖而产生流水线停顿



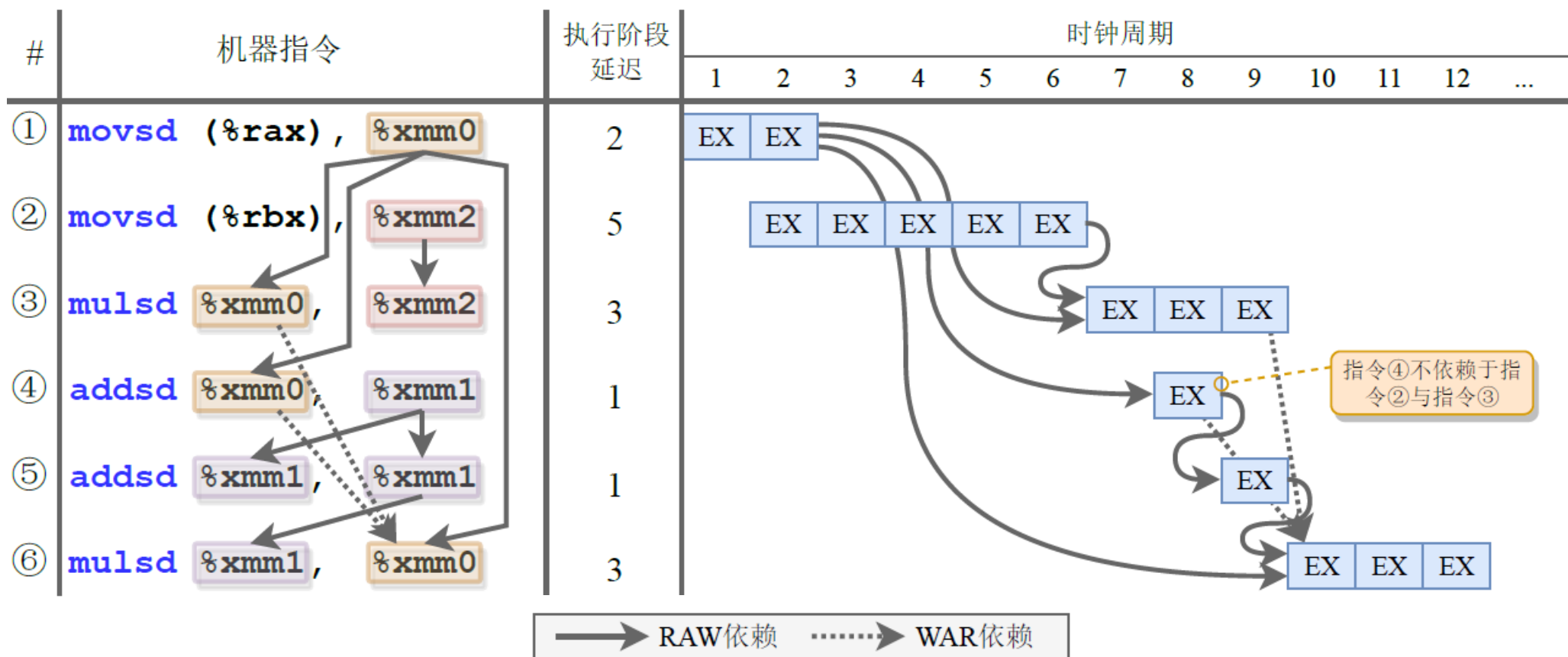
(b) 使用旁路技术后, 部分流水线停顿可以被消除



(c) 旁路技术无法避免所有流水线停顿

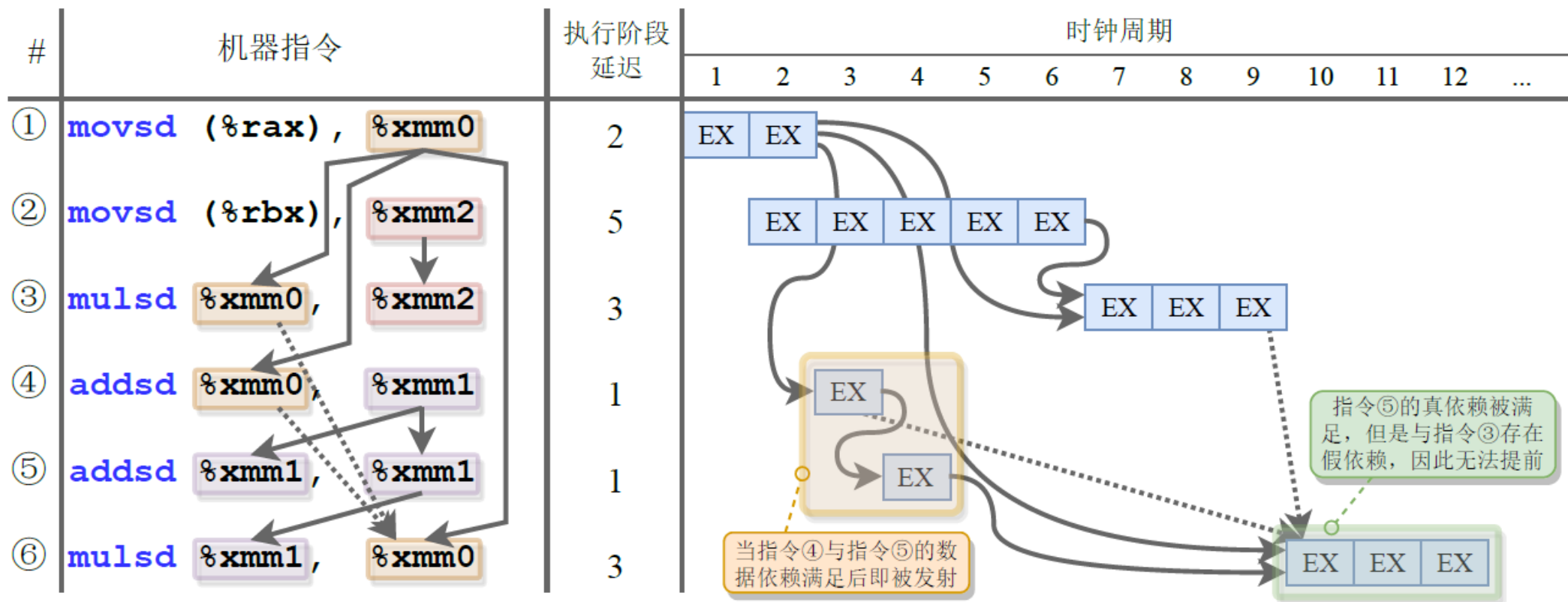
# 顺序执行

- 假设流水前端在每一个时钟周期仅向后端发射一条指令，忽略除执行阶段外的其他阶段，且执行阶段有足够多的执行单元并行地执行



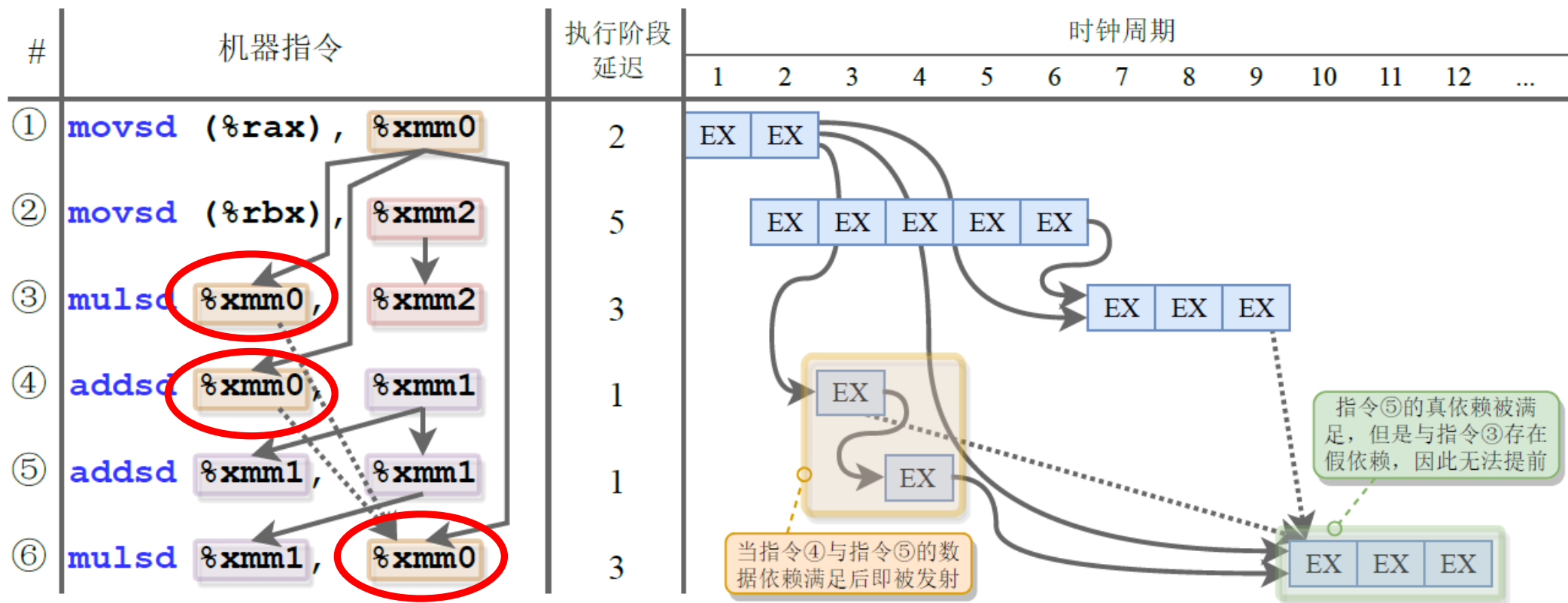
# 乱序执行

- 假设指令的数据依赖满足后就立刻发射并进入执行阶段
- 尽管指令执行的顺序被打乱，但指令间数据依赖关系没有变化



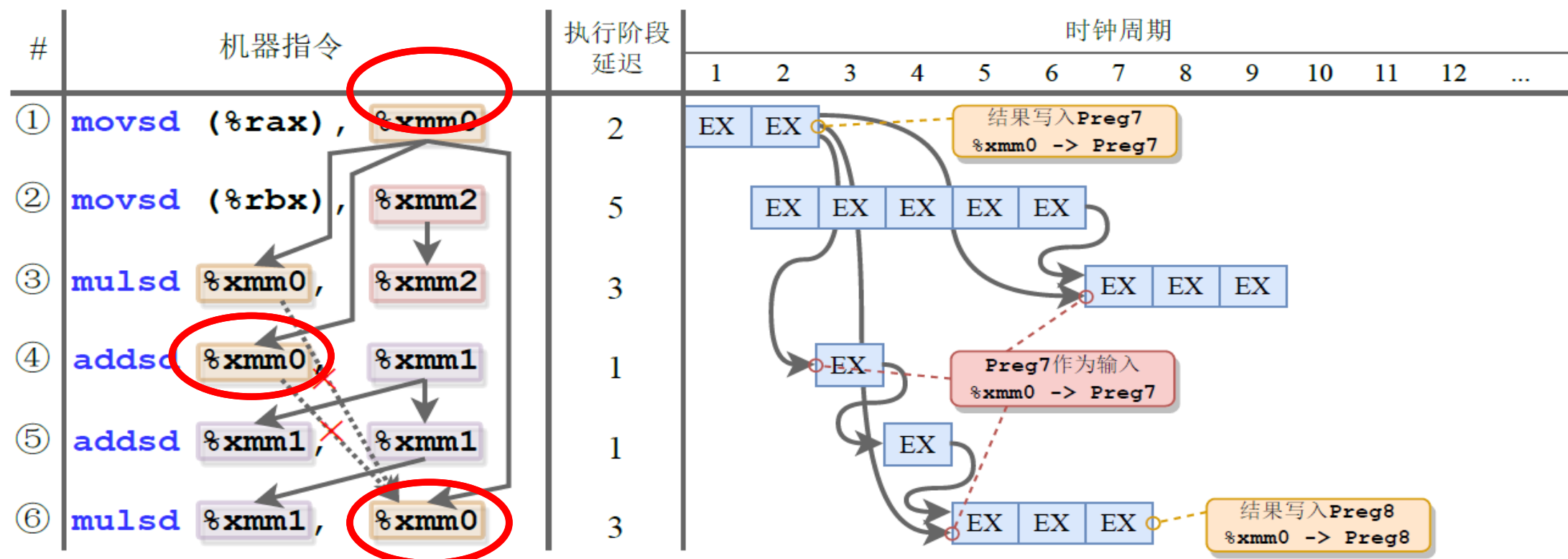
# 乱序执行

- 问题：多条指令使用同一寄存器而产生的资源竞争
- 假如有额外的临时寄存器存放 %xmm0 变化前后的值？



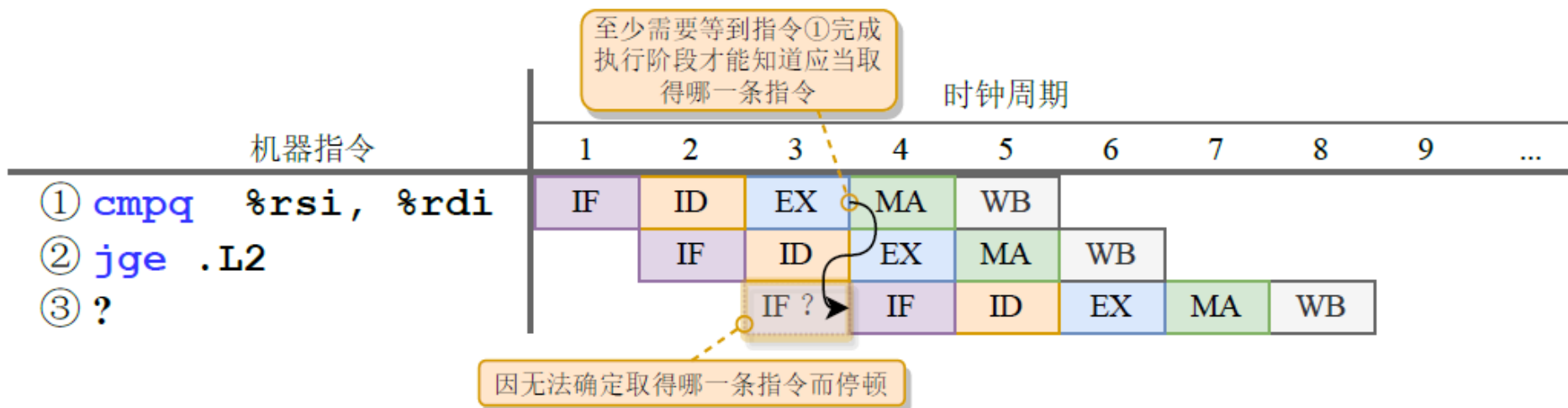
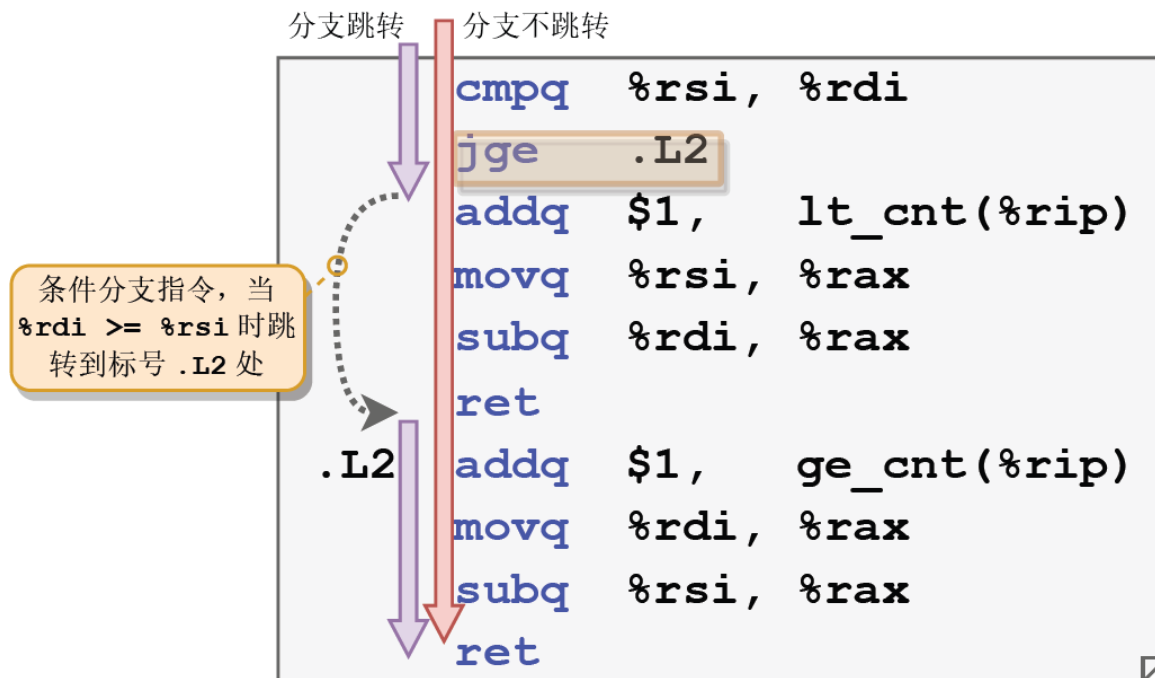
# 乱序执行 + 寄存器重命名

- 在硬件层面，处理器实现了一组额外的物理寄存器，这些寄存器并非指令集架构定义的。当指令需要使用某个指令集架构定义的寄存器时，实际上将会为其分配一个空闲的物理寄存器，以确保不同指令之间不会发生冲突。



# 推测执行

- 条件分支造成的控制冒险



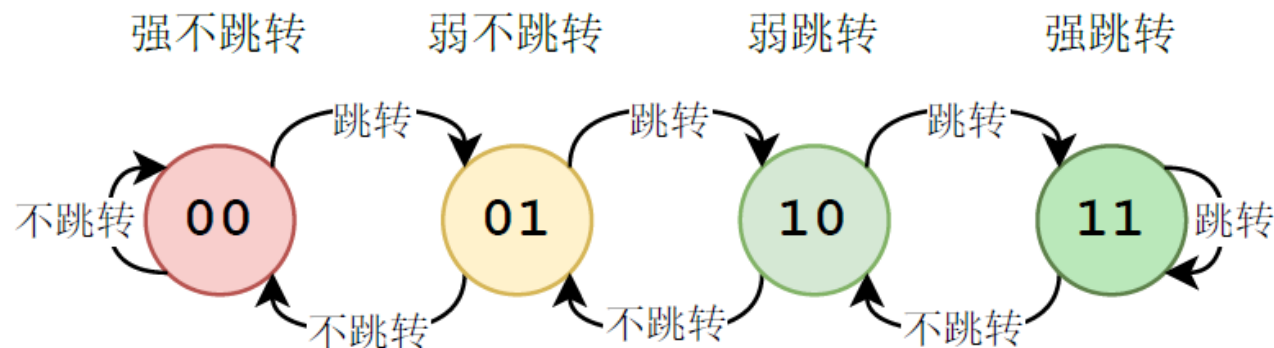
# 推测执行 / 分支预测

- 当指令流水线碰到条件分支时，提前预测条件分支是否跳转，按照预测的指令流继续执行，而不是等到跳转条件确定后再继续执行，那么，当跳转条件判断完成后：
  - 如果发现推测正确，流水线继续前进，不会发生停顿。
  - 如果发现推测错误，此时流水线将停止错误分支指令流的执行，撤销错误分支上已经执行的指令，重新执行正确分支上的指令流，这样的过程称为流水线刷新（flush）。
- 当分支预测失败时，流水线刷新的过程会使得整个指令流水线停顿，造成较大的性能损失

# 分支预测器

- 通常作用于指令流水线的取指阶段，其目的在于提高分支预测的正确率，以更加高效地实现推测执行
- 一种简单的分支预测算法是基于饱和计数器（saturating counter）或双模态预测器（bimodal predictor）的，分支预测器内部维护了一张指令跳转地址与预测结果的映射表。

分支跳转地址	预测结果
0x400c0c	01
0x400c1b	11
0x400c47	10
0x400cad	01
...	...





# 小结

- 现代处理器在微体系结构设计层面做了相应的优化设计，以提高指令级并行性
- 针对结构冒险，引入超标量指令流水线，通过设置多个执行单元和实现多发射机制，使得指令流水线的吞吐量突破 $IPC = 1$ 的限制
- 针对数据冒险，通过旁路和乱序执行机制，缓解了RAW 数据依赖造成的流水线停顿，通过重命名机制，消除了WAW 和WAR 数据依赖造成的流水线停顿
- 针对控制冒险，即为了处理程序中出现的条件分支指令，通过设置分支预测器进行推测执行，在预测正确的情况下消除了等待分支条件确定时的流水线停顿

# 扩展阅读：官方手册

- Intel® 64 and IA-32 Architectures Software Developer Manuals.  
<https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html>
- The microarchitecture of Intel, AMD, and VIA CPUs.  
<https://www.agner.org/optimize/microarchitecture.pdf>
- Intel® Intrinsics Guide.  
<https://software.intel.com/sites/landingpage/IntrinsicsGuide/>
- Arm Architecture Reference Manual Armv8.  
<https://developer.arm.com/documentation/ddi0487/latest/>
- RISC-V ISA Specifications. <https://riscv.org/technical/specifications/>