

Lecture 21: 知识 - 指令流水线

指令流水线基本概念及其性能指标

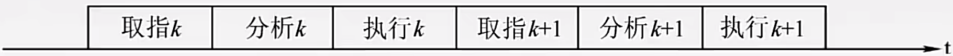
指令流水基本概念

概念：指令流水

- 概念：
 - 一条指令的执行过程可以分成多个阶段 (或过程)
- 举例：三阶段
 - 取指：
 - 根据 **PC** 内容访问主存储器，取出一条指令送到 **IR** 中；
 - 分析：
 - 对指令操作码进行译码，按照给定的寻址方式和地址字段中的内容形成操作数的有效地址 **EA**，并从有效地址 **EA** 中取出操作数；
 - 执行：
 - 根据操作码字段，完成指令规定的功能，即把运算结果写到通用寄存器或主存中；

顺序执行方式：

1. 顺序执行方式 总耗时 $T = n \times 3t = 3nt$



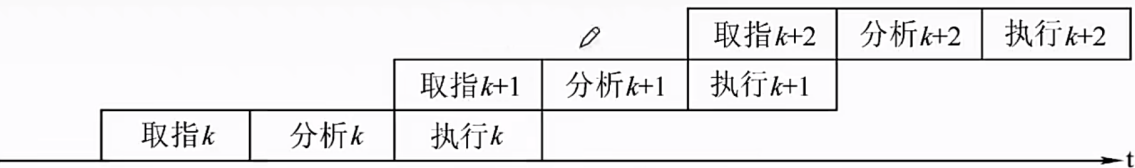
传统冯·诺依曼机采用顺序执行方式，又称串行执行方式。

优点：控制简单，硬件代价小。

缺点：执行指令的速度较慢，在任何时刻，处理机中只有一条指令在执行，各功能部件的利用率很低。

一次重叠执行方式：

2. 一次重叠执行方式 总耗时 $T = 3t + (n-1) \times 2t = (1+2n)t$

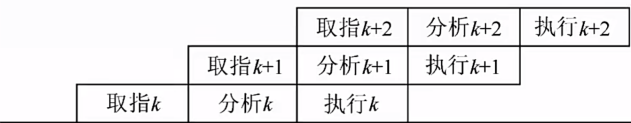


优点：程序的执行时间缩短了1/3，各功能部件的利用率明显提高。

缺点：需要付出硬件上较大开销的代价，控制过程也比顺序执行复杂了。

二次重叠执行方式：

3. 二次重叠执行方式 总耗时 $T = 3t + (n-1) \times t = (2+n)t$



与顺序执行方式相比，指令的执行时间缩短近2/3。这是一种理想的指令执行方式，在正常情况下，处理机中同时有3条指令在执行。

概念：流水线 CPU 时钟周期

公式：

- CPU 时钟周期 = MAX(各部件所需时间) + 流水段寄存器延时；

• 分析：

• 题目：

【2018 统考真题】若某计算机最复杂指令的执行需要完成 5 个子功能，分别由功能部件 A~E 实现，各功能部件所需时间分别为 80ps、50ps、50ps、70ps 和 50ps，采用流水线方式执行指令，流水段寄存器延时为 20ps，则 CPU 时钟周期至少为 ()。

- A. 60ps B. 70ps C. 80ps D. 100ps

• 分析：

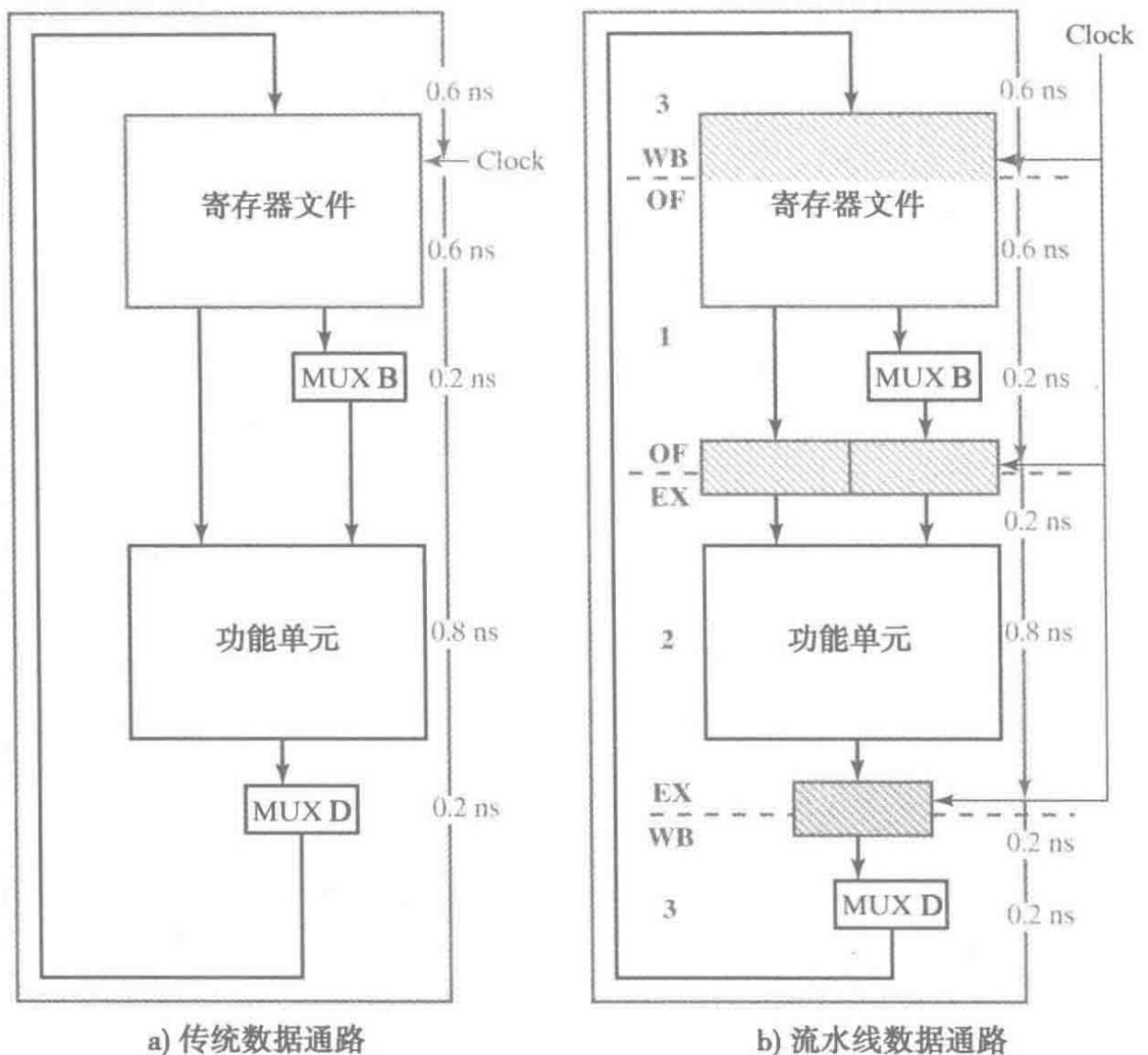
- 各功能部件所花时间为 80, 50, 50, 70, 50，其中最大的是 80；
- 流水段寄存器延时为 20；
- 因此 CPU 时钟周期至少为 80+20=100

补充：流水线模式下数据通路时序图

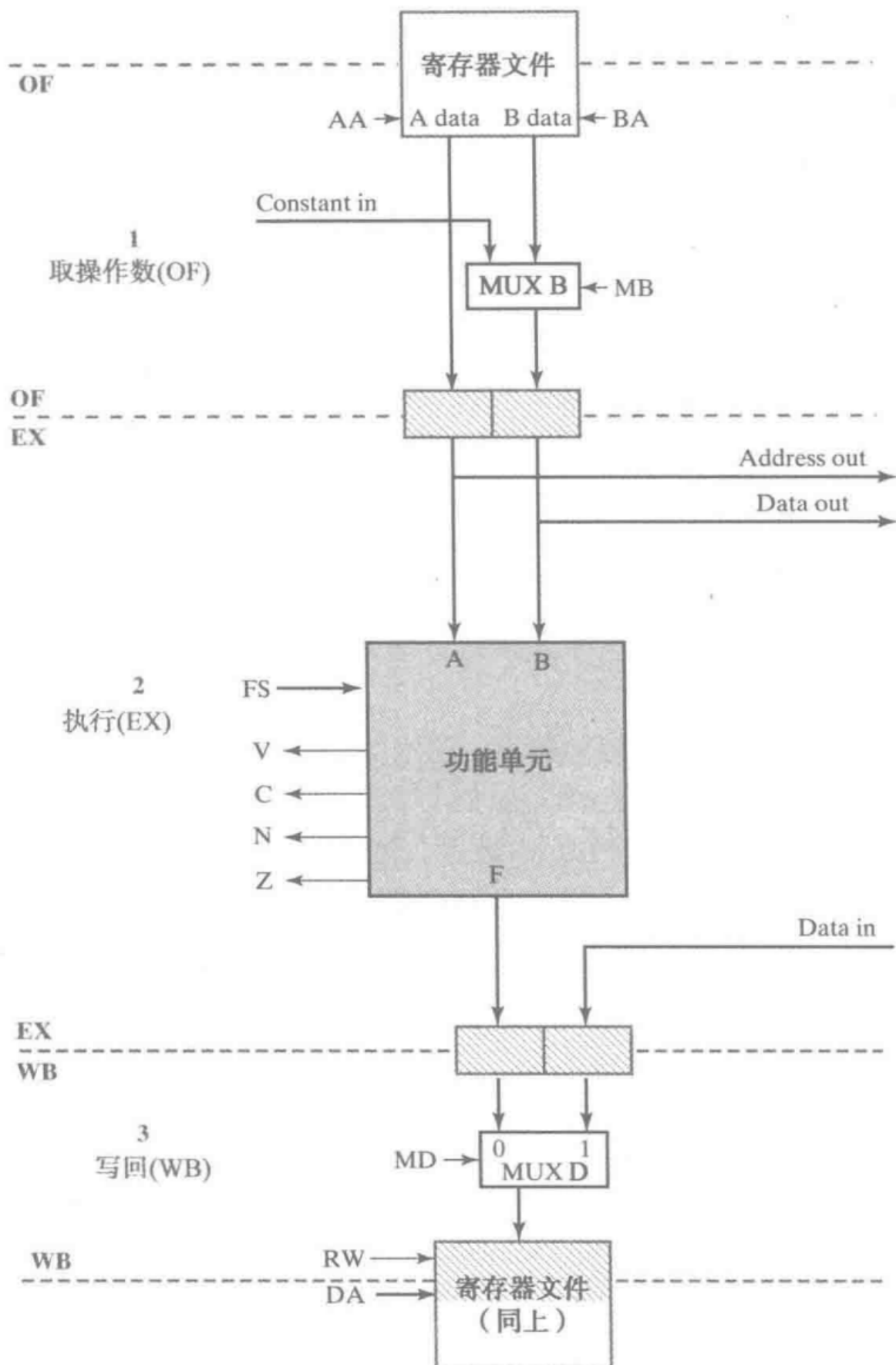
• 概念：流水线寄存器

- 为了分离流水线各段，在流水线的各段间插入寄存器，这些能够临时保存流水线中数据的数据寄存器称为流水线站；

• 图示：

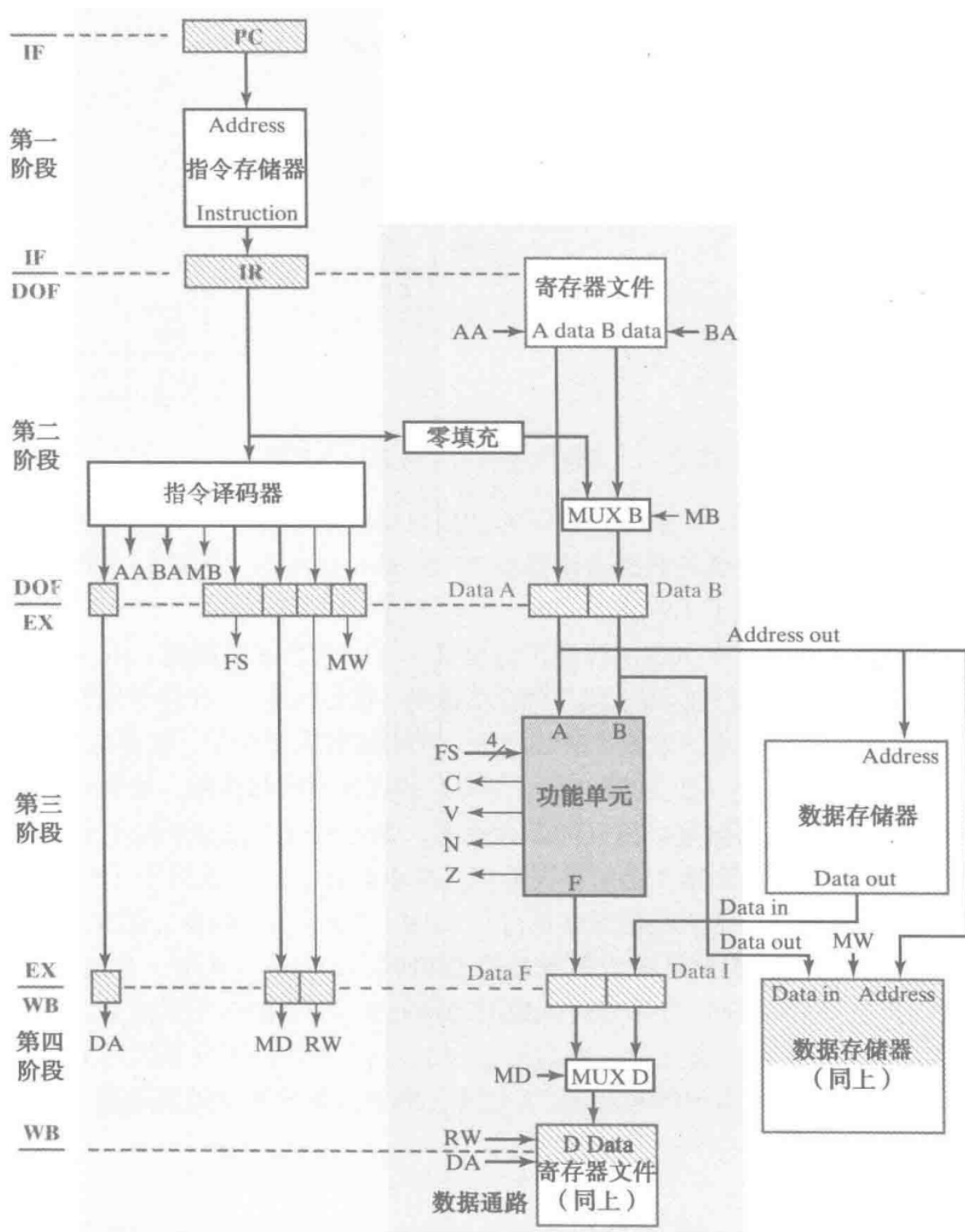


• 图示：流水线数据通路



补充：流水线控制框图

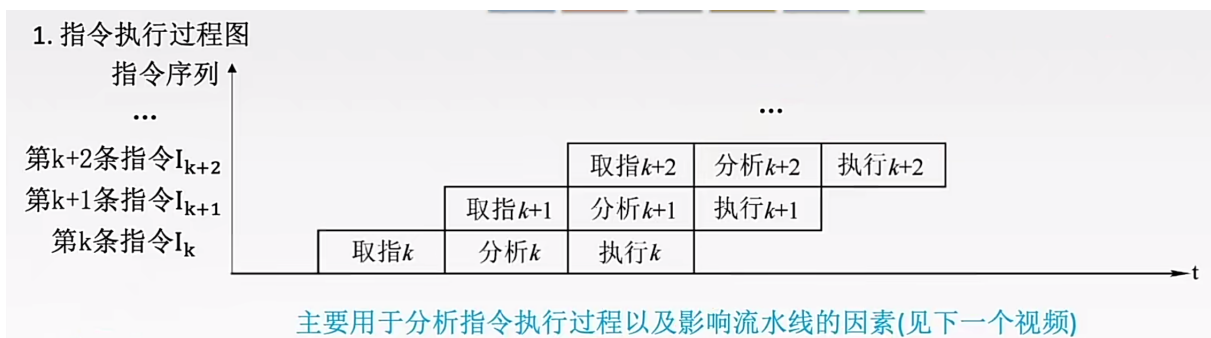
• 图示：



流水线表示方法

概念：流水线表示方法

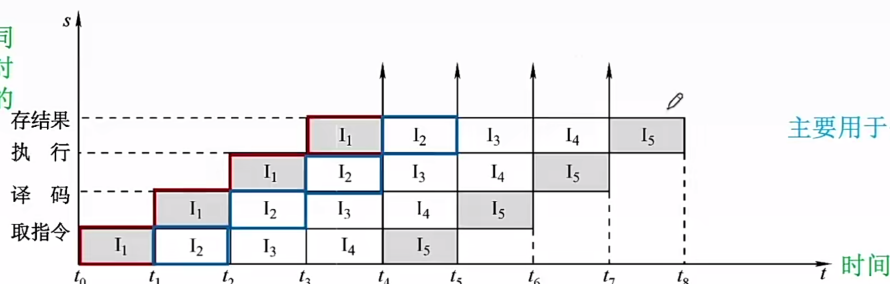
- 1. 指令执行过程图



2. 时空图

2. 时空图

空间：不同的阶段所对应的不同的硬件资源



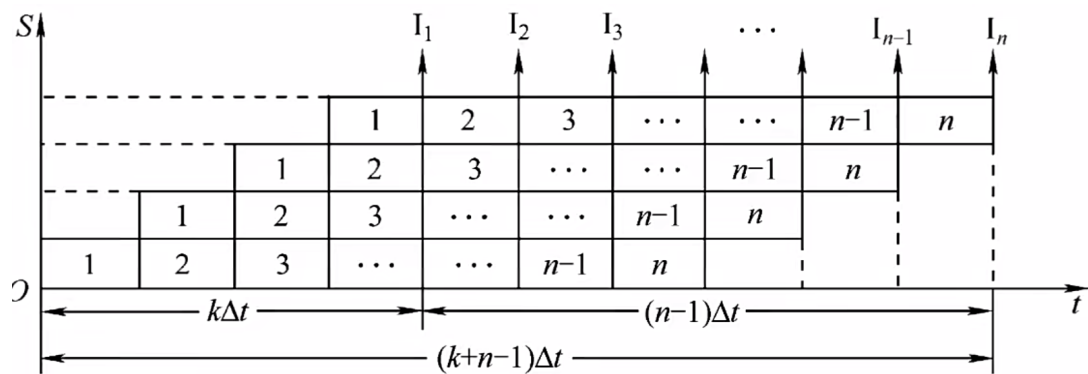
流水线的性能指标

概念：指令流水线的有利因素

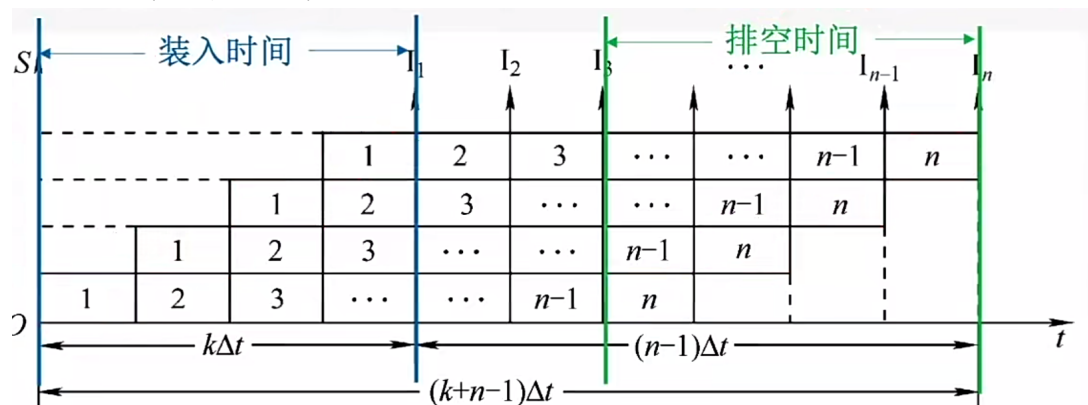
- **真题：** 2011年
- (1) 指令格式规整且长度一致；
 - 指令长度应尽量一致，这样有利于简化取指令和指令译码操作。否则会使指令部件极其复杂，取指令所花时间也长短不一；
- (2) 指令和数据按边界对齐存放；
 - 数据和指令在存储器中要“对齐”存放。这样有利于减少访存次数，使所需数据在一个流水段内就能从存储器中得到；
- (3) 指令格式应尽量规整，尽量保证源寄存器的位置相同；
 - 这样有利于在指令未知时就可取寄存器操作数。否则，必须先译码后才能确定指令中各寄存器编号的位置；
- (4) 只有 **Load/Store** 可以对操作数进行存储访问；
 - 采用 **Load/Store** 型指令风格可以保证除 **Load/Store** 指令外的其他指令 (如运算指令) 都不访问存储器，这样，可把 **Load/Store** 指令的地址计算和运算指令的步骤规整在同一个周期中；

概念：流水线吞吐率

- **概念：**
 - **定义：**
 - 指在单位时间内流水线所完成的任务数量，或是输出结果的数量；
 - **时间：**
 - **图示：**



- 图示：装入时间和排空时间



- 公式

- 设任务数为 n ，处理完成 n 个任务所用的时间为 T_k

$$TP = \frac{n}{(k+n-1)\Delta t}$$

- 极限情况下，当 $n \rightarrow \infty$ 时， $TP_{\max} = \frac{1}{\Delta t}$

- 补充：

- 真题：2020年
- 理想情况下，基本流水线的 $CPI=1$

- 举例：

- 题目：

- 真题：2013年
- 某 CPU 主频为 1.03 GHz，采用 4 级指令流水线，每个流水段的执行需要 1 个时钟周期；
- 假定 CPU 执行 100 条指令，在其执行过程中，没有发生流水线阻塞，此时流水线的吞吐率为多少？

- 分析：

- 1. 确定时钟周期数

- 已知当前流水线不会发生阻塞，因此第一条指令的执行需要 4 个时钟周期，后续每条指令的前三个阶段都在前一天指令执行时间同步进行，因此后 99 条指令每条只需 1 个时钟周期；

- 即 $n_{\text{时钟周期数}} = 4 + 1 * 99 = 103$

- 2. 分析执行 100 条指令所需的时间

- 已知当前执行 100 条指令需要 103 个时钟周期；

- 当前 CPU 的主频为

$$f = 1.03\text{GHz} = \frac{1.03 * 10^9 \text{个时钟周期}}{1\text{秒}}$$

- 所以吞吐率为：频率乘以其指令数和对应的时钟周期数的比例

$$f * \frac{n_{\text{指令数}}}{n_{\text{时钟周期数}}} = 1.03 * 10^9 * \frac{100}{103} = 1.0 * 10^9 \text{指令/秒}$$

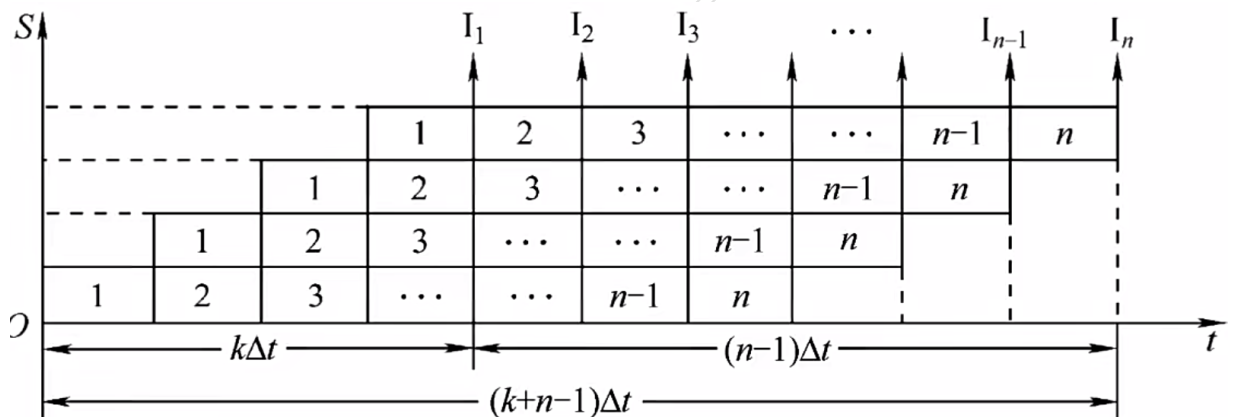
- 其含义便表示，在当前 CPU 主频下，每秒钟可以执行多少条流水线指令；

概念：加速比

- 概念：
 - 完成同样一批任务，不使用流水线所用的时间与使用流水线所用的时间之比；
- 公式：

$$S = \frac{kn\Delta t}{(k+n-1)\Delta t} = \frac{kn}{k+n-1}$$

- 图示：



概念：流水线的效率

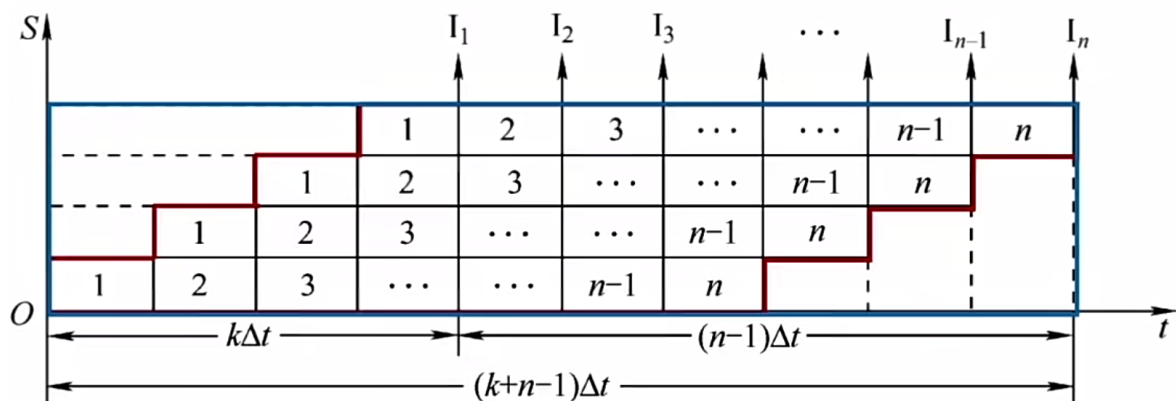
- 概念：
 - 指的就是流水线的设备利用率；
 - 在时空图上，流水线的效率定义为完成 n 个任务占用的时空区有效面积与 n 个任务所用的时间与 k 个流水段所围成的时空区总面积之比；

- 公式：

$$\text{流水线效率}(E) \text{的一般公式} \quad E = \frac{n \text{个任务占用} k \text{时空区有效面积}}{n \text{个任务所用的时间与} k \text{个流水段所围成的时空区总面积}} = \frac{1}{k}$$

- 当 $n \rightarrow \infty$ 趋向于无穷时，效率趋向于 1；

- 图示：



- 一条指令的执行分为 k 个阶段，每个阶段耗时 Δt ，一般取 $\Delta t = \text{一个时钟周期}$

流水线影响因素

五段式指令阶段简述

概念：五段式指令阶段

• 阶段：

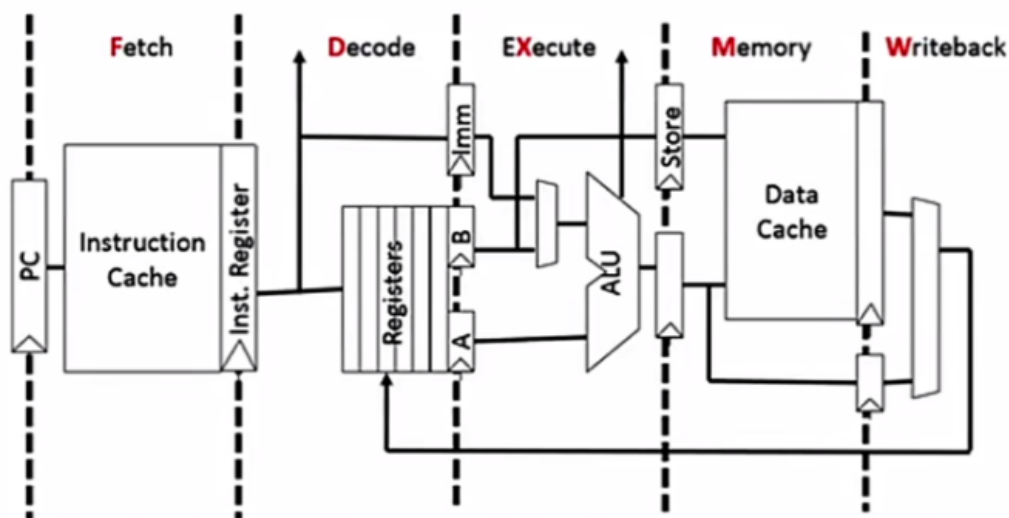
- IF: Instruction Fetch -> 取指令
- ID: Instruction Decode -> 指令阶码
- EX: Execute -> 执行
- M: Memory -> 访存
- WB: Writeback -> 写回

• 概念：

- 1. 为方便流水线的设计，将每个阶段的耗时设成一样，以最长耗时为准；
 - 即如果最大耗时为 `100 ns`，此处应将机器周期设置为 `100 ns`；
- 2. 流水线缓冲寄存器：
 - 流水线每一个功能段部件后面都要有一个缓冲或称为锁存器寄存器，其作用是保存本流水段的执行结果，提供给下一流水段使用；
 - 又称之为**锁存器**或**流水线站**；
 - 这样可以将每一段所花的时间维持一致；
- 3. Instruction Cache
 - 把指令缓存在 `Cache` 中，大部分情况下都不需要在内存中获得指令，而是可以命中 `Cache`；
- 4. Data Cache：
 - 给数据一个缓存，这样可以使得指令和数据的处理并行进行；
- 5. 在 `ID` 的阶段当中，除了要进行指令译码的操作，还需要进行**取数**；
 - 比如：`ADD` 操作的操作数直接来自主存，但是在 `RISC` 中、数据都是直接来自寄存器当中，如果需要访问数据，都需要先放在寄存器当中，然后再读取使用；
 - 因此，在 `ID` 阶段当中，还需要在通用寄存器当中、取出其所需要的操作数，放入 `A` 和 `B` 的锁存器当中；
- 6. 在 `EX` 阶段当中，有时候处理的数据可能直接来自于 `IMM`，即来自于立即数、而不是寄存器；

- 也有可能一个来自立即数、一个来自寄存器，或者都来自寄存器；
- 7. 在 WB 阶段当中，可能需要把得到的数据、写回到某个寄存器当中，参与 ID 阶段的计算；

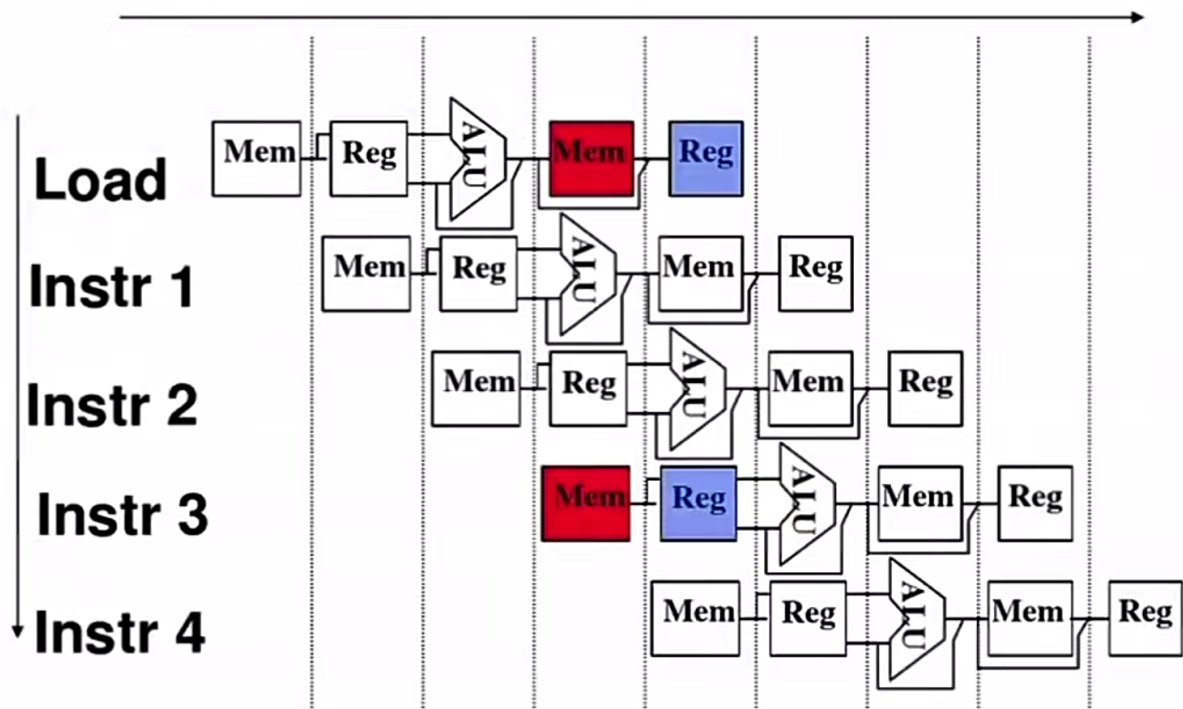
• 图示：



影响因素：结构相关 - 资源冲突

概念： 资源冲突

- 概念：
 - 由于多条指令在同一时刻争用同一资源而形成的冲突，称为结构相关；
- 方法：
 - 核心：处理互斥问题
 - 资源不够，一个人等待，或者多出来一个资源；
 - 1. 后一相关指令暂停一周期；
 - 2. 资源重复配置：数据存储器 + 指令存储器；
- 举例：
 - 两条指令在同一时刻同时访问 Mem 资源；
- 图示：



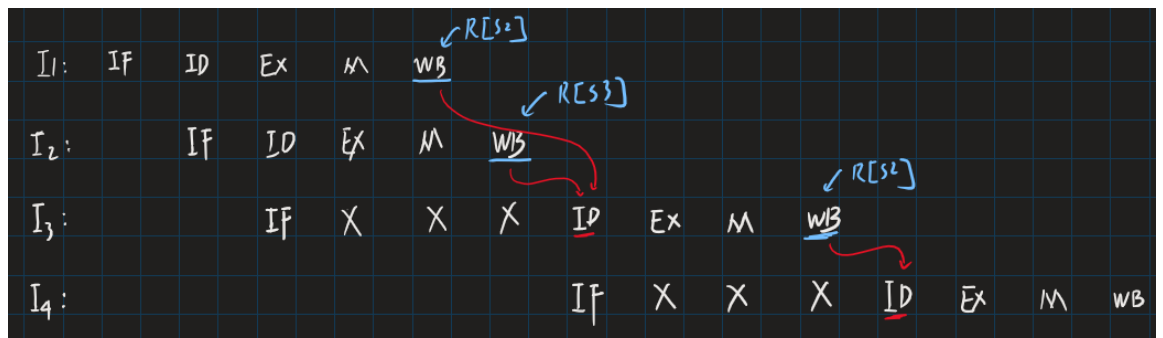
影响因素：数据相关 - 数据冲突

概念：数据冲突

- 概念：
 - 数据相关指在一个程序中，存在必须等前一条指令执行完、才能执行后一条指令的情况，则这两条指令即为数据相关；
 - 真题：2024年
 - 产生数据冒险时，只要产生数据冒险的指令延迟若干个时钟周期，就一定可以解决数据冒险，因此可以插入 nop 指令。并且通过将与前一条无数据相关的指令调整到其后执行，也可以解决数据相关；
- 分析：
 - 题目：

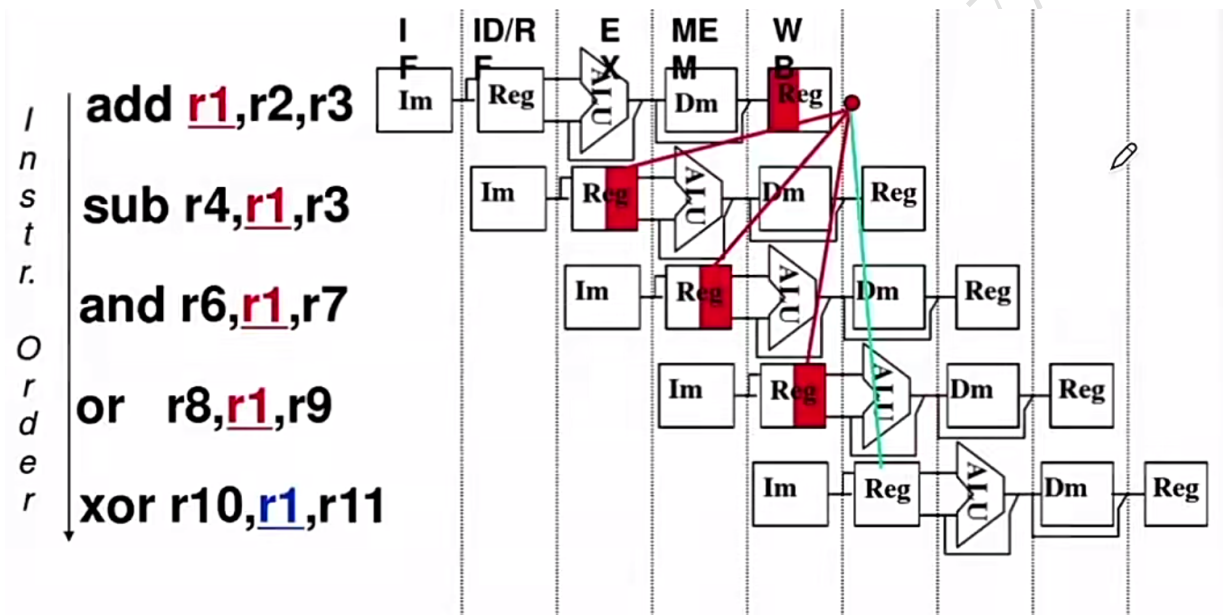
【2019 统考真题】在采用“取指、译码/取数、执行、访存、写回”5段流水线的处理器中，执行如下指令序列，其中 s0、s1、s2、s3 和 t2 表示寄存器编号。

I1: add s2, s1, s0	// $R[s2] \leftarrow R[s1] + R[s0]$
I2: load s3, 0(t2)	// $R[s3] \leftarrow M[R[t2] + 0]$
I3: add s2, s2, s3	// $R[s2] \leftarrow R[s2] + R[s3]$
I4: store s2, 0(t2)	// $M[R[t2] + 0] \leftarrow R[s2]$
 - 下列指令对中，不存在数据冒险的是 ()。
- 解析：



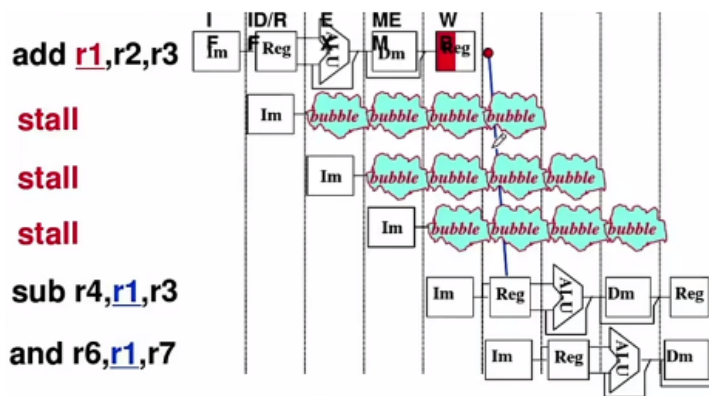
- 答案:
 - I₁ 和 I₂、I₃ 产生数据冒险;
 - I₃ 和 I₄ 产生数据冒险;
 - I₂ 和 I₄ 不产生数据冒险;

• 图示:

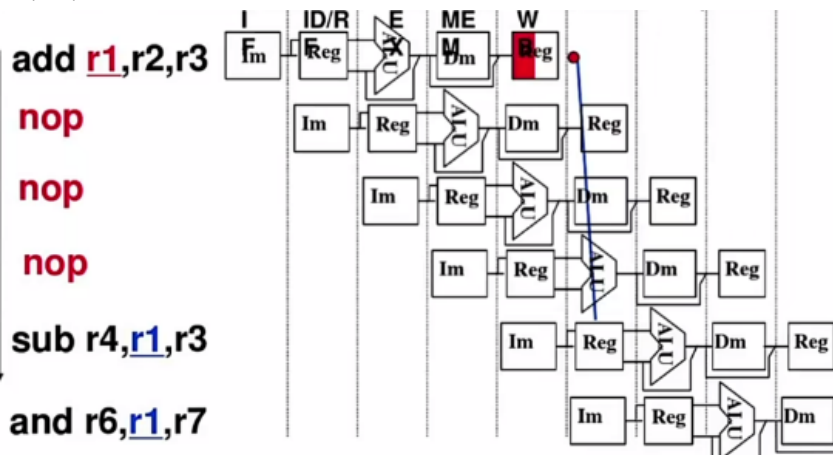


方法: 数据冲突的三种处理方法

- 1. 等待:
 - 概念:
 - 把遇到数据相关的指令及其后续指令都暂停一至几个时钟周期, 直到数据相关问题消失后再继续执行;
 - 可分为硬件阻塞 `stall` 和软件插入 `NOP` 两种方法;
 - 举例: 硬件阻塞
 - 硬件插入空拍, 即 `bubble`



- 举例：软件插入

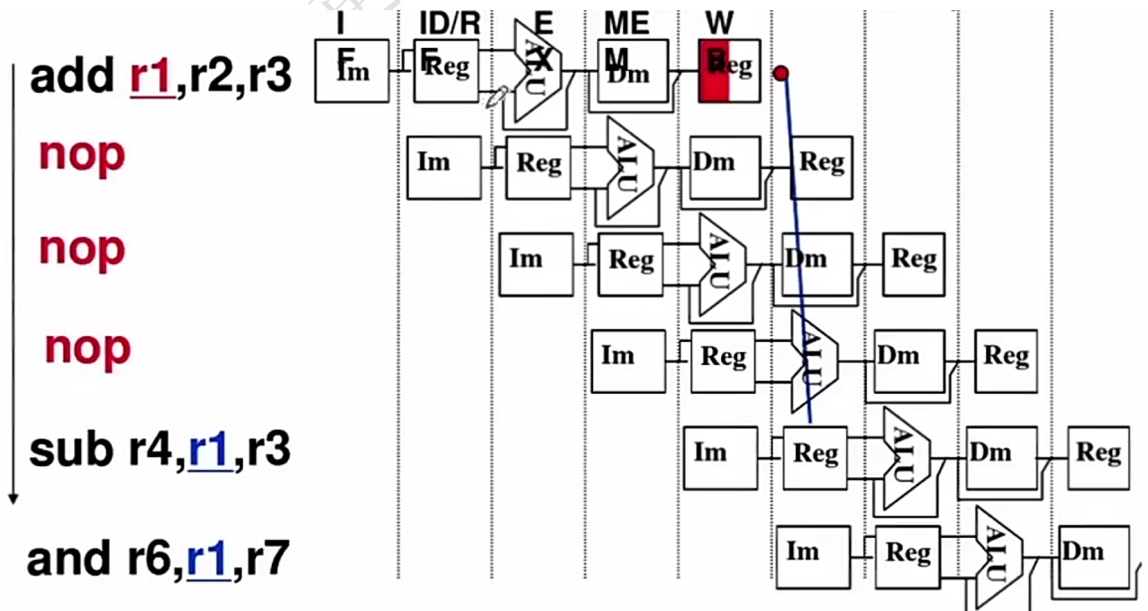


- 2. 数据旁路技术:

- 概念:

- 在 ALU 的输出端连接一条数据线路，使其直接与 ALU 的输入端相连接，此时便可以在 EX 阶段之和完之和、其他指令就可以使用到此数据；

- 图示:



- 补充：数据旁路技术不能处理所有的数据冲突

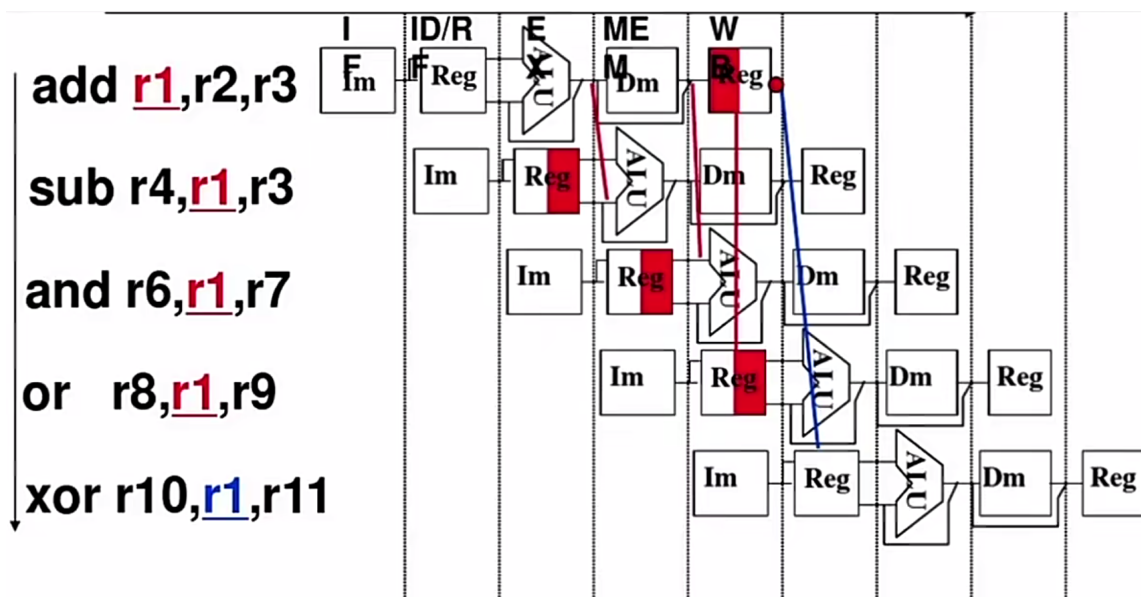
- 真题：2024年

- load-use 型数据冒险，通过转发电路不能解决，依然要延迟一个时钟周期；

- 3. 编译优化技术:

- 概念:

- 通过编译器调整指令顺序来解决数据相关问题；
- 可以把无依赖性的指令提前到有依赖性指令之前、被依赖的指令之和；
- 图示：

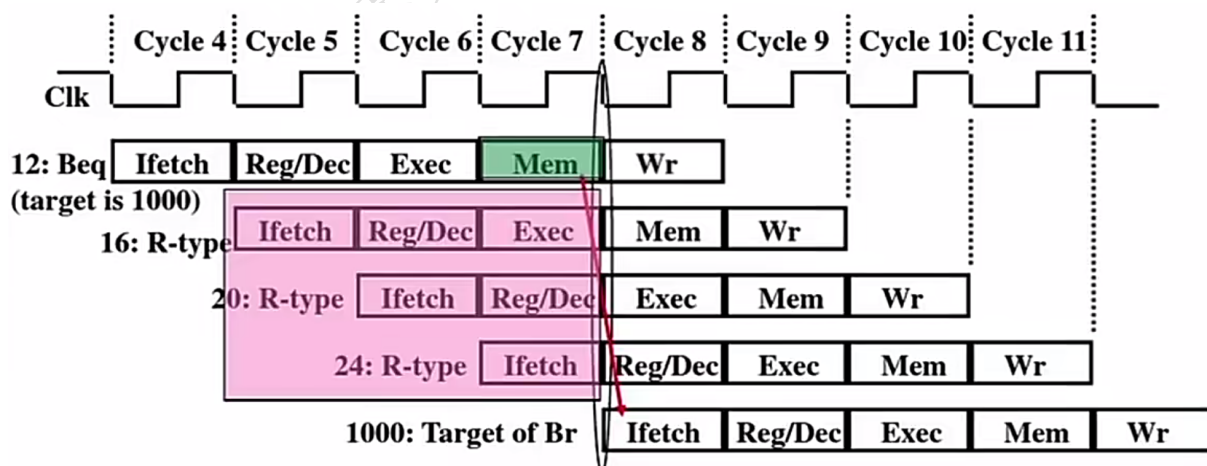


影响因素：控制相关 - 控制冲突

概念：控制冲突

- 概念：
 - 当流水线遇到转移指令和其他改变 PC 值的指令而造成断流时，会引起控制相关；
 - 比如：本来在 `if` 的指令执行顺序执行、不跳过时，会执行括号里面的指令。但如果此时不符合条件，则会跳过当中部分指令，此时提前读出来的一些指令就会被跳过执行流；

- 图示：



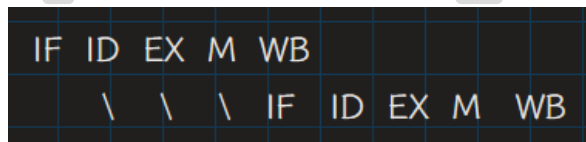
方法：控制冲突处理方法

- 1. 转移指令分支预测
 - 1. 简单预测：
 - 概念：
 - 永远猜正确或错误；
 - 2. 动态预测：

- 预提取出成功以及不成功的两个控制流方向上的目标指令；
- 2. 加快和提前形成条件码；
- 3. 预取转移成功和不成功两个控制流方向上的目标指令；
- 4. 提高转移方向的猜准率；

分析：分支指令引起的控制冒险

- 真题：2014年
- 控制相关的解决
 - 从流水线上来看，计算转移的目的地址放在 EXE 阶段，判断是否能成功转移，也是放在 EXE 阶段；
 - 控制冲突简单解决方法：一旦发现分支指令，就暂停流水线，即暂停该指令之后的所有指令，直到分支指令达到 MEM 段确定了新的 PC 值为止；
 - 暂停几个周期，可以根据操作码知道；
 - 因此：假设 bne 发生控制相关，可以将其下一条指令的 IF 阻塞，一直阻塞到 M 之后；
 - 因为在 M（访存）时，会更新当前的 PC 值；



- 总结：
 - bne 指令的 PC 写回发生在 M 阶段
 - jmp 指令的 PC 写回发生在 EXE 阶段

分析：中断指令引起的控制冒险

- 除了由于分支指令引起的控制冒险外，还有异常或中断引起的控制冒险；
- 异常和中断的出现会改变程序的执行流程，使得流水线执行发生阻塞；
- 与分支相比，当某条指令执行过程中发现异常或中断时，可能它后面的多条指令已经被取到流水线正在执行；
 - 例如：ALU 运算类指令发现“溢出”时，已经执行到 EXE 阶段，此时，它后面已两条指令进入了流水线；
 - 通过在数据通路的不同流水段中加入相应的检测逻辑可检测出哪条指令发生。例如，“溢出”可在 EXE 段检出；
 - “无效指令”可在 ID 段检出；
 - “除数为 0”可在 ID 段检出；
 - “无指令地址”可在 IF 段检出；
 - “无效数据地址”可在 load/store 指令的 EXE 段或 MEM 段检出检测；
 - 出异常的那个流水段正在执行的指令，就是发生异常的指令
- 外部中断的检测，可以放在第一个流水段的 IF 或最后一个流水段 WB 中进行。若放在 IF 中检测，因为可在取指令前进行，若发现有中断请求发生，则能确保在该时钟周期就开始执行中断服务程序；若在 W 阶段进行，则需要将刚执行完的指令后面几条指令从流水线中清除；

指令流水线的分类

概念：指令流水线的分类 - 基于流水线的使用级别不同

- 分类：
 - 部件功能级；
 - 处理机级；
 - 处理机间级流水线；
- 概念：
 - 根据流水线使用的级别的不同，流水线可分为部件功能级流水线、处理机级流水线和处理机间流水线；
- 解释：
 - 1. 部件功能级流水：
 - 概念：
 - 就是将复杂的算术逻辑运算组成流水线工作方式；
 - 比如把 ALU 部件内的功能再划分为几个指令（子过程）
 - 举例：
 - 可将浮点加法操作分成求阶差、对阶、尾数相加以及结果规格化等 4 个子过程；
 - 2. 处理机级：
 - 概念：一条指令解释过程分成多个子过程，如前面提到的取指、译码、执行、访存及写回 5 个子过程；
 - 3. 处理机间级流水线
 - 概念：
 - 处理机间流水是一种宏流水，其中每一个处理机完成某一专门任务，各个处理机所得到的结果需存放在与下一个处理机所共享的存储器中；

概念：指令流水线的分类 - 基于单/双功能流水线不同

- 分类：
 - 单功能流水线；
 - 多功能流水线；
- 概念：
 - 单功能流水线：
 - 指只能实现一种固定的专门功能的流水线；
 - 多功能流水线：
 - 指通过各段间的不同连接方式，可以同时或不同时地实现多种功能的流水线；

概念：指令流水线的分类 - 基于动态/静态流水线不同

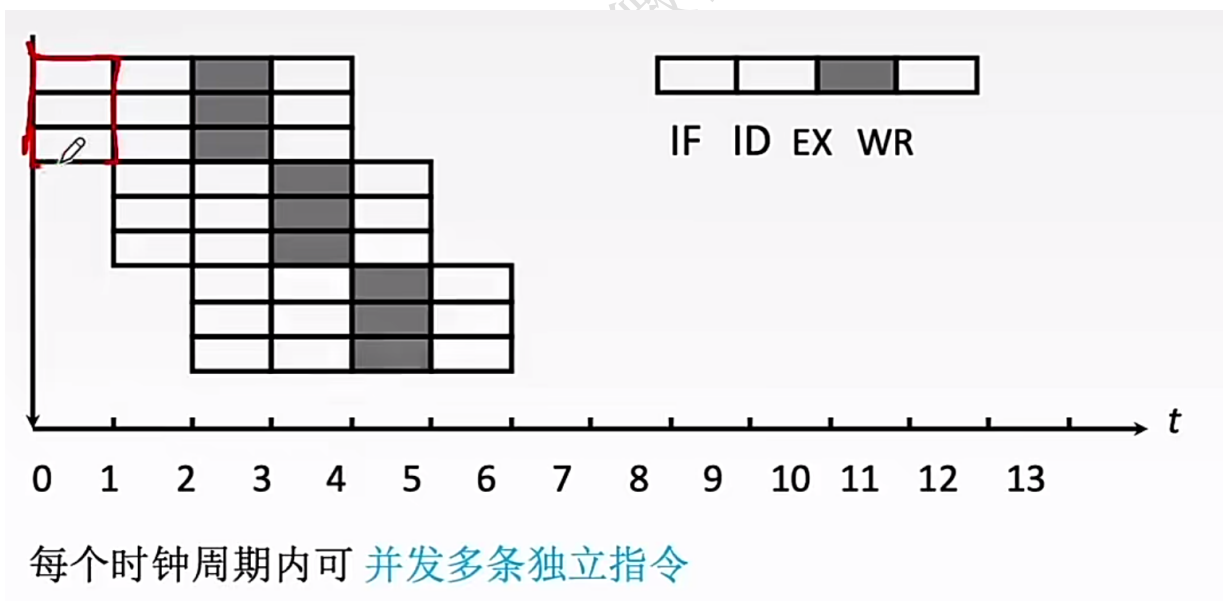
- 分类：
 - 动态流水线；
 - 静态流水线；
- 概念：
 - 按同一时间内各段之间的连接方式，流水线可分为静态流水线和动态流水线；
- 解释：

- 静态流水线：
 - 指在同一时间内，流水线的各段只能按同一种功能的连接方式工作；
- 动态流水线：
 - 指在同一时间内，当某些段正在实现某种运算时，另一些段却正在进行另一种运；
 - 这样，对提高流水线的效率很有好处，但会使流水线控制变得很复杂；

流水线的多发技术

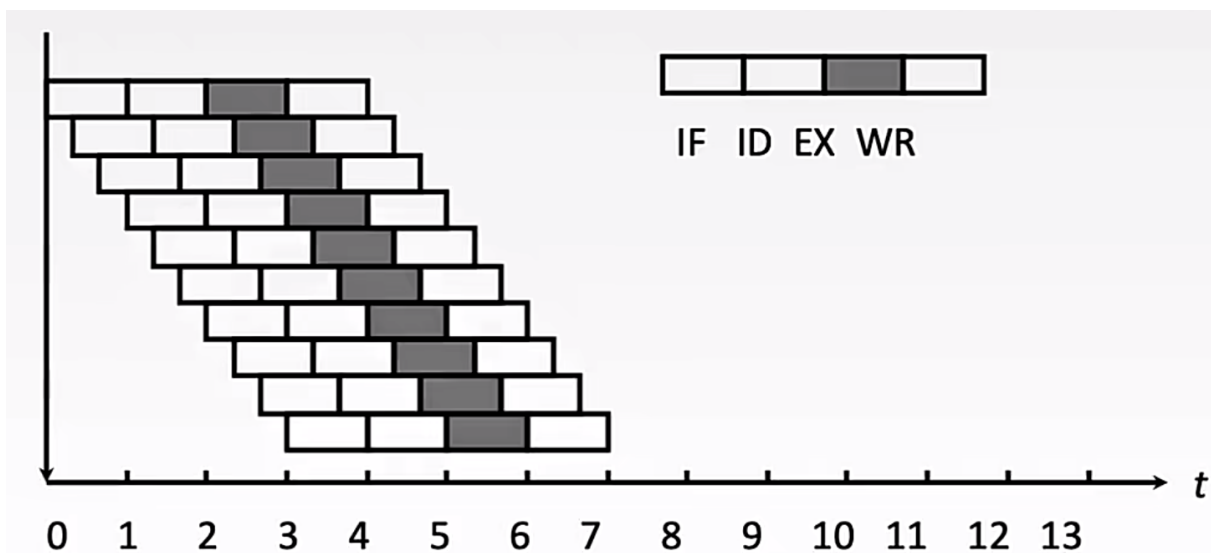
概念：超标量技术

- 概念：
 - 每个时钟周期内可**并发多条独立指令**；
 - 需要保证几条同时运行的指令，需要同时配置多个**功能部件**；
 - 不能随便调整指令的执行顺序；
 - 指令的排列完全由编译器来进行，因此对于编译优化的要求高；
- 特点：
 - 真题：2017年
 - (1) 能在一个时钟周期发射多条指令；
 - (2) 能结合**动态调度**技术提供指令执行的并行性；
- 图示：



概念：超流水技术

- 概念：
 - 在一个时间周期内再分段，再同一个时间周期内、不同的时间点发出多个指令；
 - 因此在一个时间周期当中，一个功能部件使用多次；
 - 不能随便调整指令的执行顺序；
 - 指令的排列完全由编译器来进行，因此对于编译优化的要求高；
- 图示：



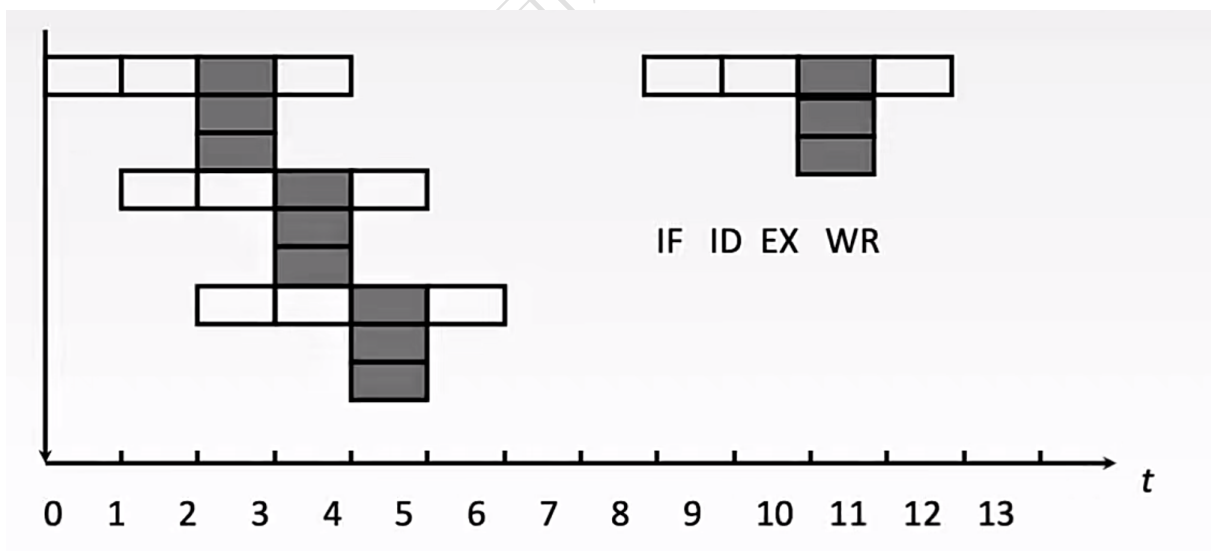
在一个时钟周期内再分段（3段）

概念：超长指令字

• 概念：

- 当编译程序发现不同的指令有可以并行执行的部分时，可以把并行的操作部分组合为一条，在执行阶段的时候再进行分开执行；
- 具有多个操作码字段的超长指令字；
- 具有多个操作码字段的超长指令字（可达几百位）；

• 图示：



五段式流水线指令结构

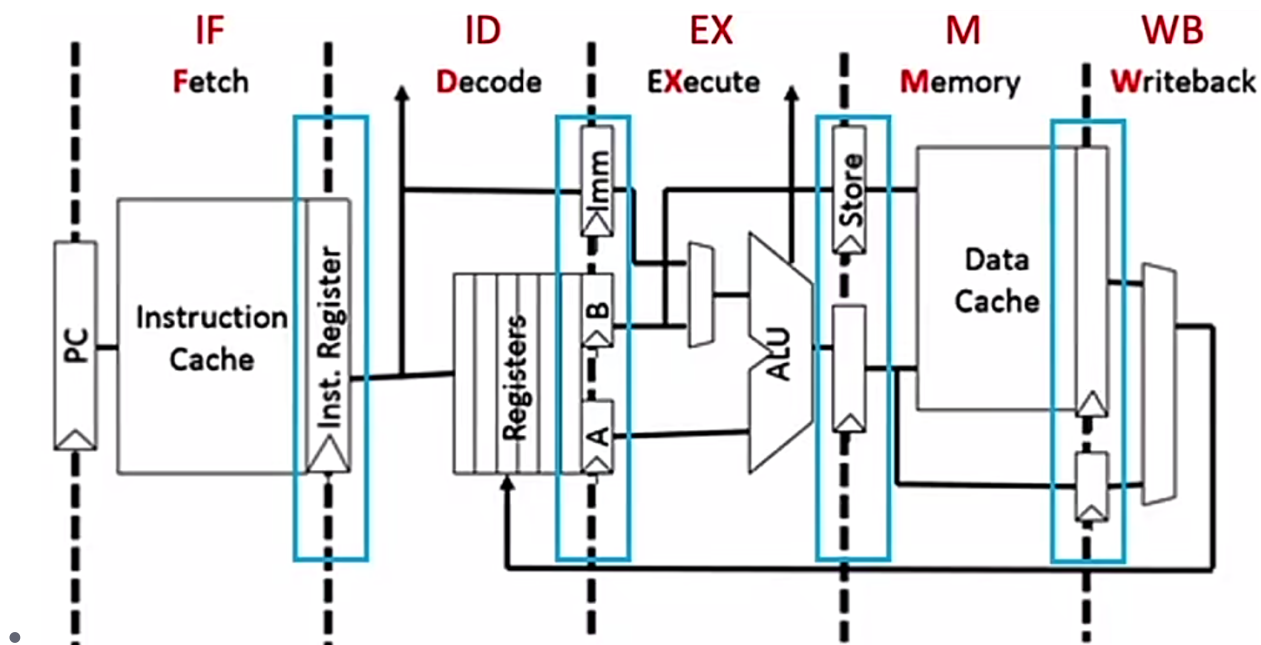
基础概念

概念：五段式流水线指令结构

• 过程：

- ①IF 取指 > ②ID 译码&取数 > ③EX 执行 > ④M 访存 > ⑤WB 写回寄存器；

• 图示：



运算类指令

举例： ADD Rs, Rd

- 解释：
 - 加法指令，两个寄存器中的数字相加；
- 功能：
 - $(Rs) + (Rd) \rightarrow Rd$

举例： ADD #966, Rd

- 解释：
 - 加法指令，取立即数；
- 功能：
 - $996 + (Rd) \rightarrow Rd$

举例： SHL Rd

- 解释：
 - 算数左移指令；
- 功能：
 - $(Rd) \ll 2 \rightarrow Rd$

分析： 运算类指令执行过程分析

- 1. IF
 - 根据 PC 从指令 Cache 取指令至 IF 段的锁存器；
 - 注意：当前指令一般是在指令 Cache 中，因此 IF 的取指令操作涉及到访存，只不过一般直接在指令 Cache 中命中；
- 2. ID
 - 取出操作数，放至 ID 段的锁存器当中；

- 比如：ADD #966,Rd 的立即数会被放到 ID 阶段的 Imm 当中，然后把另一个操作数从寄存器读到 ID 锁存器当中；
- 3. EX
 - 运算，将结果存入 EX 段锁存器；
- 4. M
 - 如果是纯运算类指令，不需要把算出来的结果放回主存；
 - 但因为五指令流水线结构当中，要求每一段的时长对齐。因此 M 阶段是空段，时间和其他阶段一致；
- 5. WB
 - 将运算结果写回指定寄存器；

LOAD 指令

举例：LOAD Rd,996(Rs)

- 功能：
 - $(996+(Rs)) \rightarrow Rd$
- 解释：
 - 在 Rs 寄存器中取出的数据的基础之上，加上 996 $\rightarrow$ 往后偏移 996，取到 Rd 寄存器当中；

举例：LOAD Rd,mem

- 功能：
 - $(mem) \rightarrow Rd$

分析：LOAD 指令执行过程分析

- 1. IF
 - 根据 PC 从指令 Cache 取指令至 IF 段的锁存器；
- 2. ID
 - 将基址寄存器当中的值放入锁存器 A，将偏移量放入 Imm；
- 3. EX
 - 运算，得到有效地址；
- 4. M
 - 从数据 Cache 中取数，并放回锁存器当中；
 - 补充：主存里面的数据大概率会在 Cache 中命中，不命中时在进入主存访存；
- 5. WB
 - 将取出的数写回寄存器；
- 补充：
 - 在 RISC 指令集结构中，只有 LOAD 和 STORE 指令可以对主存进行访存；

STORE 指令

举例：STORE Rs,996(Rs)

- 功能:
 - $Rs \rightarrow (996 + (Rd))$
- 解释:
 - 把 Rs 指明的寄存器中的内容, 存放以 Rd 为基地址、偏移 996 的位置;

举例: $LOAD\ Rs, mem$

- 功能:
 - $Rs \rightarrow (mem)$
- 解释:
 - 把数据存入主存;

分析: $STORE$ 指令执行过程分析

- 1. IF
 - 根据 PC 从指令 $Cache$ 取指令至 IF 段的锁存器;
- 2. ID
 - 将基址寄存器当中的值放入锁存器 A , 将偏移量放入 Imm , 将要存放的数放入 B 中;
- 3. EX
 - 根据基地址和偏移量, 计算出有效地址, 并将锁存器的内容放到锁存器 $STORE$ 中;
- 4. M
 - 将数据写入 $Cache$ 中;
- 5. WB
 - 空段;
- 补充:
 - 在 $RISC$ 指令集结构中, 只有 $LOAD$ 和 $STORE$ 指令可以对主存进行访存;

条件转移类指令

举例: $beq\ Rs, Rt, \#偏移量$

- 功能:
 - 若 $(Rs) == (Rt)$, 则 $(PC) + 指令字长 + (偏移量 \times 指令字长) \rightarrow PC$; 否则 $(PC) + 指令字长 \rightarrow PC$;
- 解释:
 - 如果相等, 则执行偏移, 否则只是 PC 更改、顺序执行;

举例: $bne\ Rs, Rt, \#偏移量$

- 功能:
 - 若 $(Rs) \neq (Rt)$, 则 $(PC) + 指令字长 + (偏移量 \times 指令字长) \rightarrow PC$; 否则 $(PC) + 指令字长 \rightarrow PC$;
- 解释:
 - 如果不相等, 则执行偏移, 否则只是 PC 更改、顺序执行;

分析: BNE 指令执行过程分析

- 1. IF

- 根据 **PC** 从指令 **Cache** 取指令至 **IF** 段的锁存器；
- 2. **ID**
 - 进行比较的两个数放入锁存器 **A、B**；
 - 偏移量放入 **Imm**；
- 3. **EX**
 - 运算，比较两个数；
- 4. **M**
 - 将目标 **PC** 值写回 **PC**；
 - 注意：**PC** 不是通用寄存器，因此其操作不是在 **WB** 阶段进行，而是在 **M** 阶段；
- 5. **WB**
 - 空段；

无条件转移类指令

举例： `jmp #偏移量`

- 功能：
 - $(PC) + \text{指令字长} + (\text{偏移量} \times \text{指令字长}) \rightarrow PC$ ；
- 解释：
 - 相对于当前 **PC** 的值进行偏移；

分析： **JMP** 指令执行过程分析

- 1. **IF**
 - 根据 **PC** 从指令 **Cache** 取指令至 **IF** 段的锁存器；
- 2. **ID**
 - 偏移量放入 **Imm**；
- 3. **EX**
 - 直接根据偏移量，将目标 **PC** 值写回 **PC**；
 - 因为条件转移类指令可能会产生控制冲突，因此把其放到 **EX** 阶段、而不是 **WB** 阶段来进行；
- 4. **M**
 - 空段；
- 5. **WB**
 - 空段；

例题

例题. 假设某指令流水线采用“按序发射，按序完成”方式，没有采用转发技术处理数据相关，并且同一寄存器的读和写操作不能在同一个时钟周期内进行。若高级语言程序中某赋值语句为 $x=a+b$ ， x 、 a 和 b 均为int型变量，它们的存储单元地址分别表示为 $[x]$ 、 $[a]$ 和 $[b]$ 。该语句对应的指令序列及其在指令流中的执行过程如下图所示。

I1 LOAD R1, [a] M[a] → R1
I2 LOAD R2, [b] M[b] → R2
I3 ADD R1, R2 (R1) + (R2) → R2
I4 STORE R2, [x] (R2) → M[x]

则这4条指令执行过程中I3的ID段和I4的IF段被阻塞的原因各是什么？

	时间单元													
指令	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I ₁	IF	ID	EX	M	WB									
I ₂		IF	ID	EX	M	WB								
I ₃			IF				ID	EX	M	WB				
I ₄							IF				ID	EX	M	WB

I3与I1和I2存在数据相关；

I4的IF段必须在I3进入ID段后才能开始，否则会覆盖IF段锁存器的内容

获取笔记持续更新可加微信：WandD222