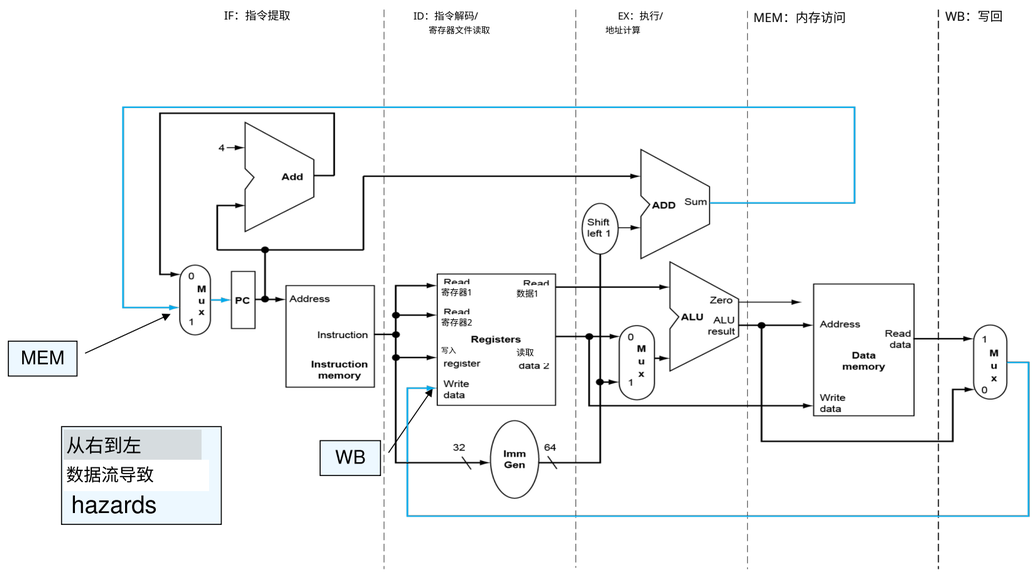
附录C

基本流水线概念与实现

1

流水线数据路径与控制



3

流水线冒险

□危害分类

>结构危害

•这些是硬件资源冲突。

>数据危害

•指令依赖于先前计算的结果
而该结果尚未准备好（计算或存储）

>控制冒险

■ 分支条件和分支程序计数器（PC）无法及时获取下一个时钟周期的指令

5

结构冒险总结

□ 冒险分类

＞ 结构冒险

•这些是硬件资源冲突。

• 好的，也许可以添加额外的硬件资源；
或者对功能单元进行全流水线处理；
否则仍需停顿

• 允许机器存在结构冒险，因为这种情况不常发生

数据冒险

• 指令依赖于尚未准备好（未计算或
尚未存储

＞ 控制冒险

•分支条件和分支 PC 未能及时获取以取下一条指令
在下一个时钟周期

7

对RISC - V指令集进行流水线处理

¬ 由于有五个独立的阶段，我们可以构建一个流水线，使每条指令处于不同阶段。

□CPI 减小到 1，因为每个周期将发出（或完成）一条指令。

□在任何周期内，每个阶段都有一条指令。

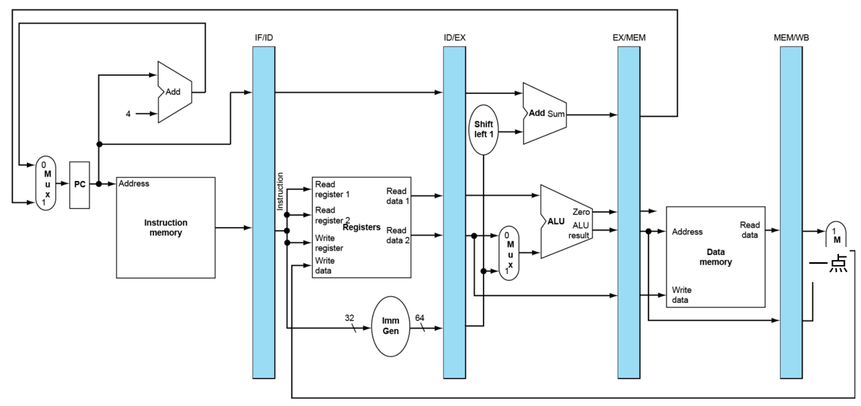
	时钟编号								
	1	2	3	4	5	6	7	8	9
指令 i	IF	ID	EX	MEM	WB				
指令 i+1		IF	ID	EX	MEM	WB			
指令 i+2			IF	ID	EX	MEM	WB		
指令 i+3				IF	ID	EX	MEM	WB	
指令 i+4					IF	ID	EX	MEM	WB

理想情况下，性能提高五倍！2

五级流水线

□各级之间需要寄存器（或锁存器）

＞ 用于保存上一周期产生的信息



4

如何解决结构冒险问题

□对内存的多次访问

＞ 分离指令和数据内存 I 多内存端口/指令缓冲区

＞ 内存带宽需要提高5倍。

□对寄存器文件的多次访问

＞ 双重碰撞

□未完全流水线化的功能单元

＞ 使功能单元完全流水线化

＞ 使用多个功能单元

□实际机器常常存在结构冒险。

6

数据冒险

当流水线与顺序执行相比改变了对操作数的读写访问顺序（违反数据依赖）时，就会发生数据冒险。

□ 让我们看一个例子

加法指令：将寄存器R1和R3的值相加，结果存入R1

减法指令：用寄存器R1的值减去R5的值，结果存入R4

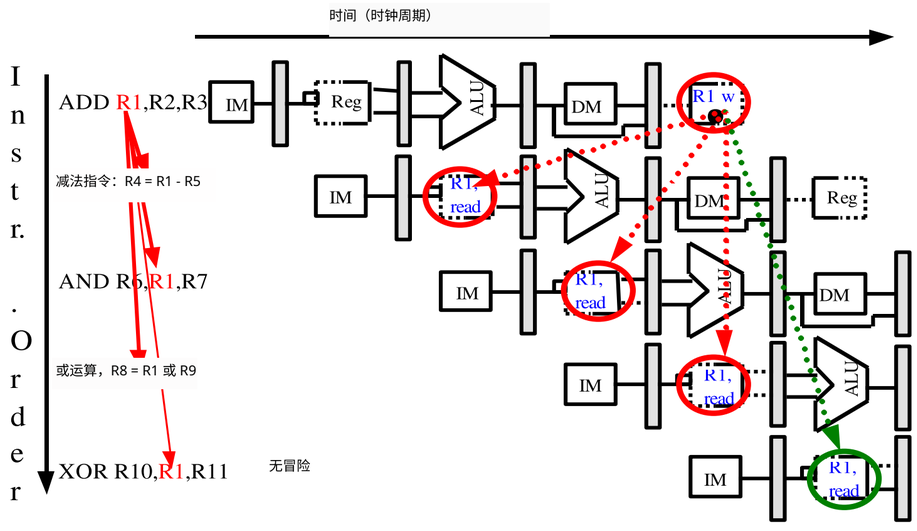
逻辑与指令：将寄存器R1和R7的值进行逻辑与运算，结果存入R6

逻辑或指令：将寄存器R1和R9的值进行逻辑或运算，结果存入R8

将R1和R11进行异或运算，结果存入R10

8

处理数据冒险：示例



数据冒险

☐ **基本结构**

> 正在执行的指令想要使用一个尚未“完成”的数据值

» “完成”意味着“已经计算出来”并且“它位于我通常期望在流水线硬件中找到它的位置” ☐ 根本原因

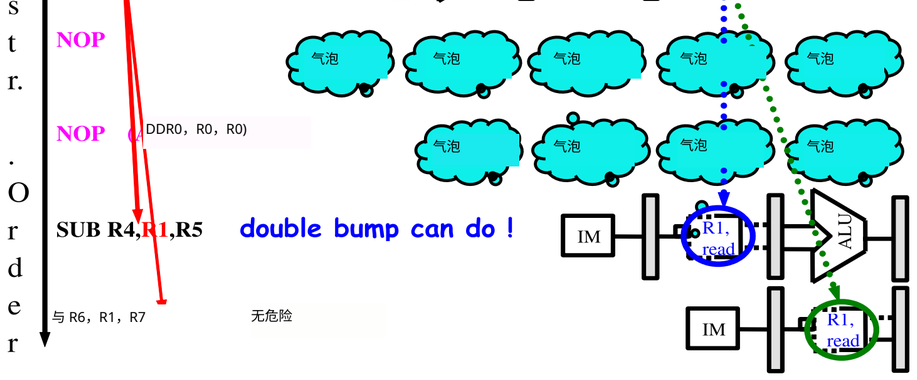
> 你习惯假设指令执行是纯顺序模型

> 指令 N 在指令 N + k 之前完成，原因是 k >= 1
> 现在“相邻”指令（按从内存中提取的顺序“相邻”）之间存在依赖关系

☐ 后果+

> 数据冒险——指令需要尚未准备好或尚未处于正确位置的数据值

某些情况下“双碰撞”可行！



解决数据冒险的最简单方法：停顿

☐ 提议的解决方案

> 别让它们像这样重叠.....?

☐ 机制

> 不要让指令流经管道
> 特别是，不要让它在代表真实CPU状态（例如，寄存器文件、内存）的管道硬件中的任何位置写入任何位

> 让指令等待，直到冒险情况得到解决。
此操作的名称：流水线停顿

我们如何进行停顿？通过编译器插入空指令

我们如何处理停顿？添加硬件互锁装置！

☐ 添加额外硬件以检测停顿情况

- 监控指令字段位
- 特别关注特定流水线阶段中的“读与写”冲突

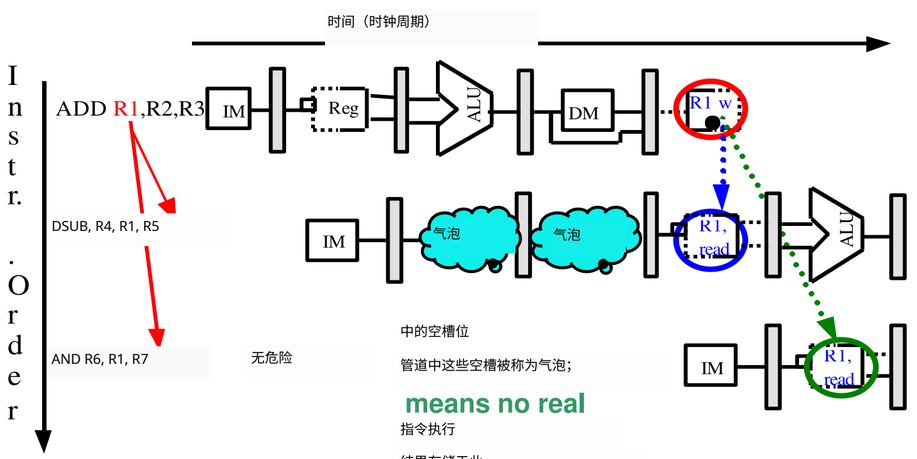
> 基本上，是一系列精心设计的“条件逻辑”

☐ 添加额外硬件以推动气泡通过流水线

> 实际上，相对容易

- 可以直接让你想要搁置的指令继续执行
通过管道.....
- .....但是，关闭允许任何结果写入机器状态
- 所以，指令“执行”(完成工作)，但不“保存”

互锁：插入停顿



联锁是如何实现的?

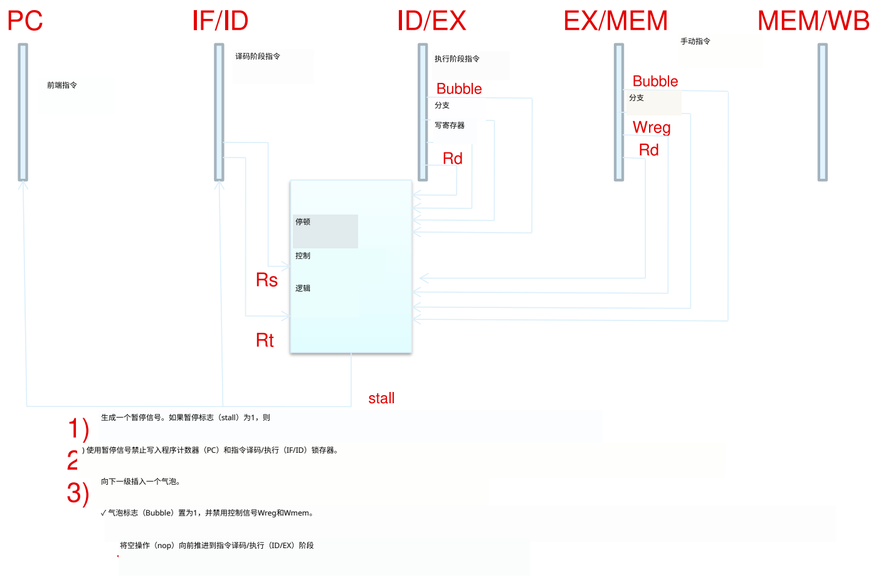
互锁功能是如何工作的?

✓互锁可以模拟空操作（NOP）：
一旦检测到需要添加停顿，那么

> 将ID/EX.IR清零，使其成为空操作指令。
•禁用写信号：“wreg，wmem”。

> 让IF/ID.IR再多保持一个时钟周期不变。
•禁用写入信号：“WritePC，WriteIR”。

如何进行停顿？添加停顿控制逻辑！



如何进行停顿（当出现数据冒险时）？

□如果停顿标志 == 0　　//停止后续操作

•然后 $\mathrm{PC} \leftarrow$ 新建 PC；　　IF/ID.IR ← IF.IR

➢　　IF/ID.NPC ← 程序计数器 + 4

数据冒险和控制冒险的停顿方式是否相同?

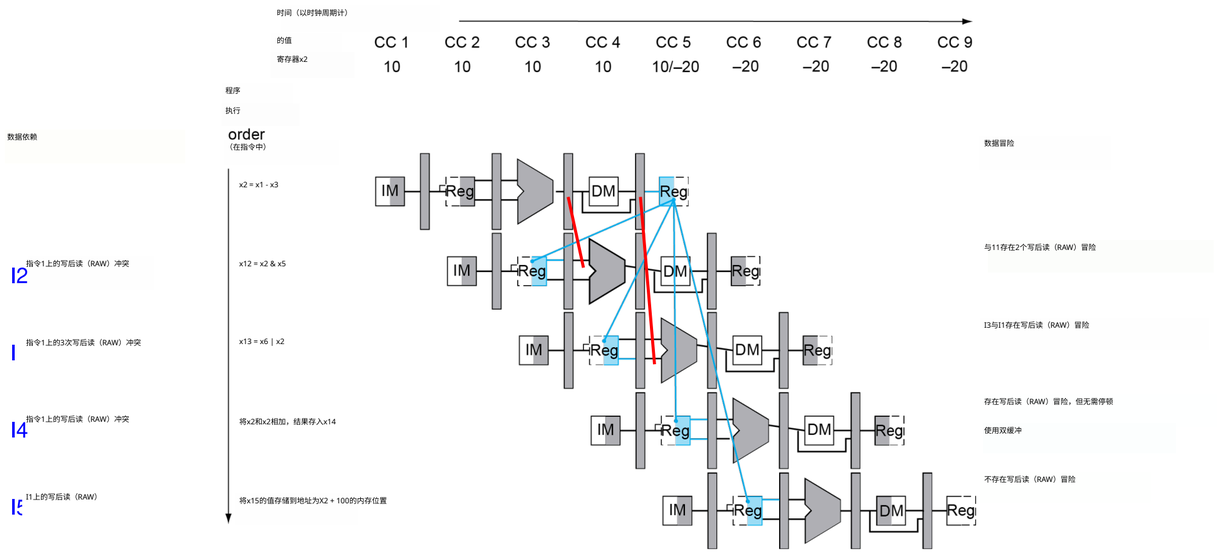
□如果停顿 == 1　　// 向前推送气泡

•那么 ID/EX.空操作 $\leftarrow 1$ 否则 ID/EX.空操作 $\leftarrow 0$

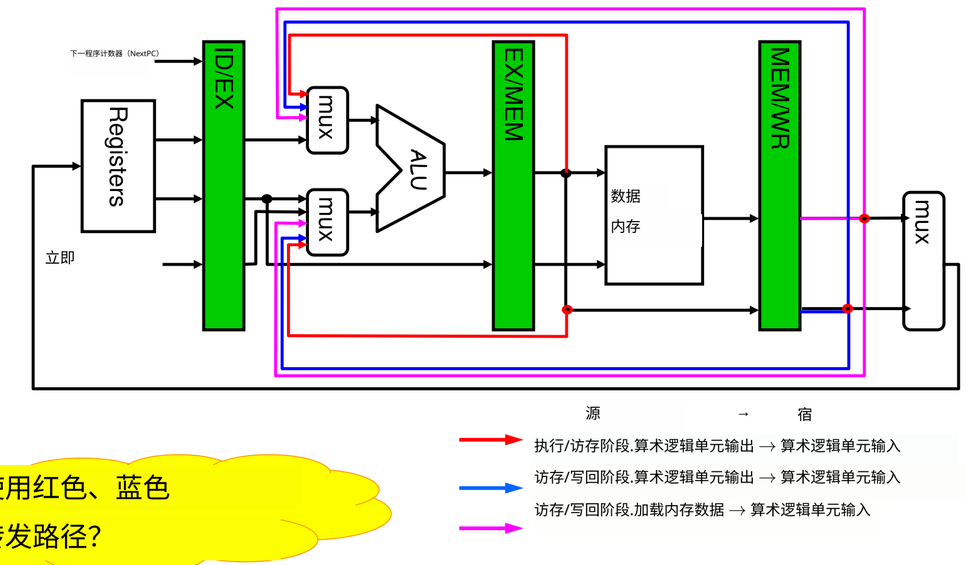
□初始时

ID/EX.nop $\leftarrow 0$，EX/MEM.nop $\leftarrow 0$，Mem.WB.nop $\leftarrow 0$

通过转发解决数据冒险问题



转发的硬件更改



何时使用红色、蓝色紫色转发路径?

何时使用转发路径?

•EX/Mem阶段的ALU输出 → ALU输入

•EX/MEM阶段的前一条指令是ALU指令

•ID/EX阶段的指令包含源寄存器Rs1或Rs2

•EX/MEM阶段的目标寄存器Rd等于ID/EX阶段的源寄存器Rs1，或者EX/MEM阶段的目标寄存器Rd等于ID/EX阶段的源寄存器Rs2

- **MEM/WB.ALUoutput → ALU input**

•MEM/WB阶段的前一条指令是算术逻辑单元（ALU）指令

•ID/EX阶段的指令包含源寄存器Rs1或Rs2

•MEM/WB阶段的目标寄存器Rd等于ID/EX阶段的源寄存器Rs1，或者MEM/WB阶段的目标寄存器Rd等于IF/ID阶段的源寄存器Rs2

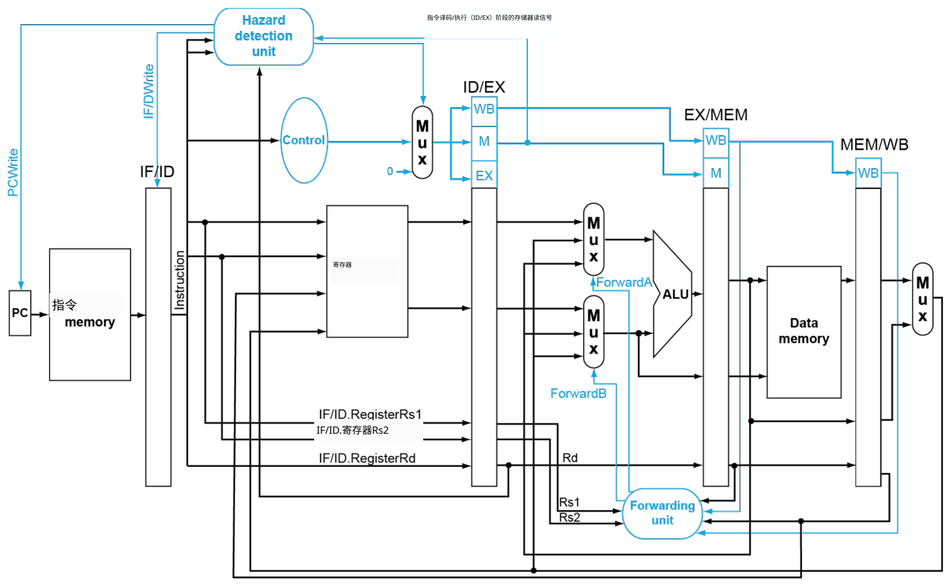
- **MEM/WB.LMD → ALU input**

•MEM/WB阶段的前一条指令是加载指令

•ID/EX阶段的指令有Rs1或Rs2源寄存器

•MEM/WB阶段的Rd寄存器等于ID/EX阶段的Rs1寄存器，或者MEM/WB阶段的Rd寄存器等于ID/EX阶段的Rs2寄存器

在执行（EX）阶段进行转发



转发只能解决部分问题

将x28和x29相加，结果存入x6

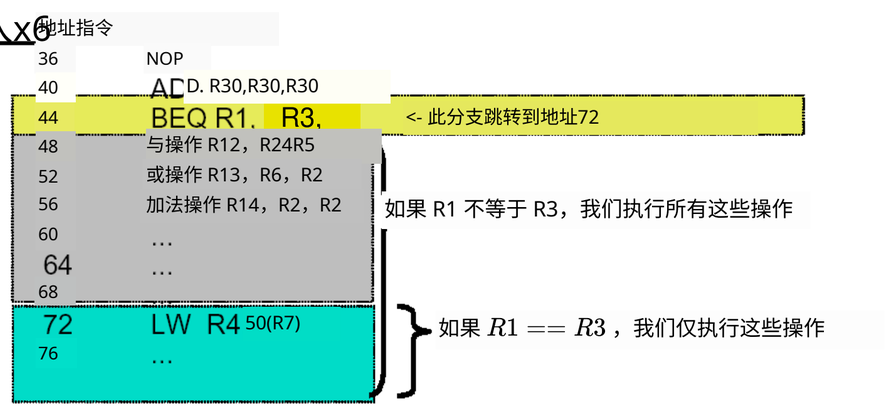
用x6减去x14，结果存入x30

将x28和x29相加，结果存入x6 / 从地址x10 + 4处加载数据存入x6

与运算 x14, x5, x7

减法运算 x30, x6, x14

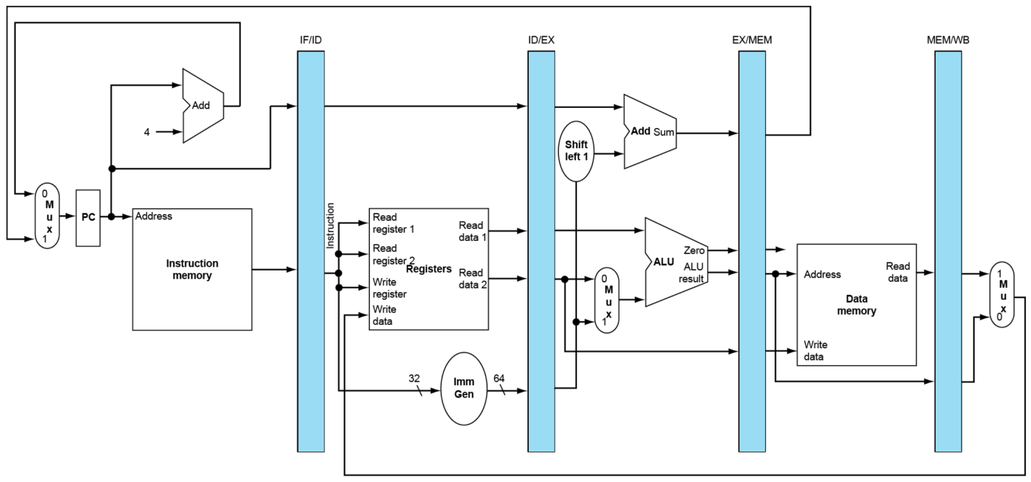
控制冒险示例：分支



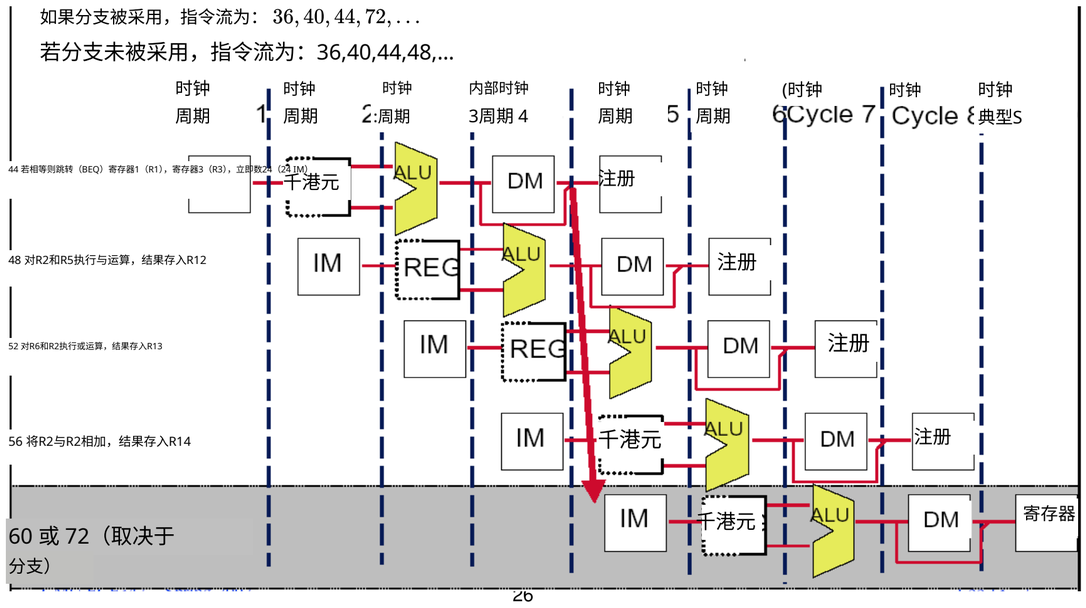
如果分支被采用，指令流为：$36, 40, 44, 72, \ldots$

如果分支未被采用，指令流为：$36, 40, 44, 48, \ldots$

回顾：基本流水线数据路径



控制冒险

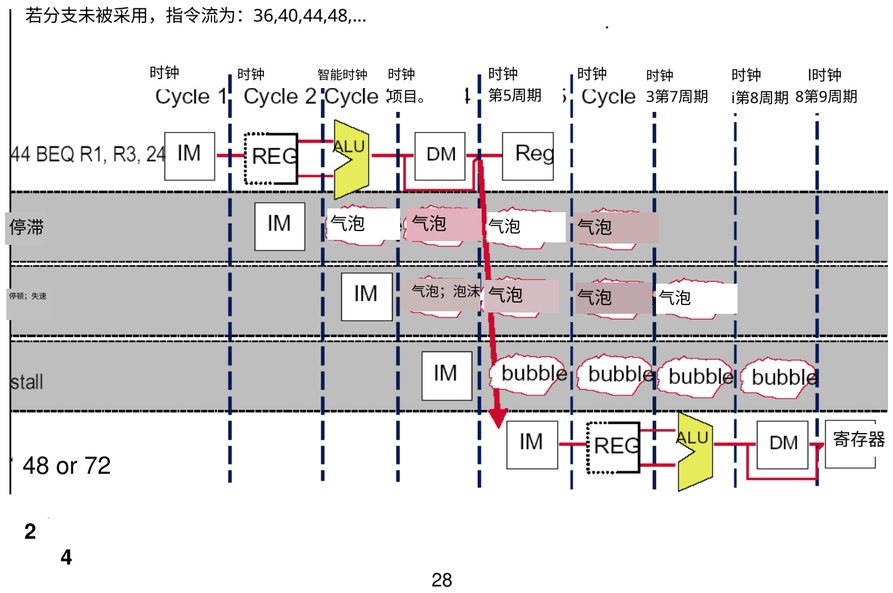


处理控制冒险

☐ 四种简单解决方案
> 停顿
> 预测
•预测不跳转：将每个分支都视为不跳转
•预测为跳转：将每个分支都视为会跳转
•延迟分支（省略）

☐ 注意：
> 固定硬件
› 利用硬件方案知识和分支行为知识的编译时方案

回顾：通过插入停顿来解决冒险问题



清空流水线

☐ 最简单的硬件：
> 在分支目标确定之前，保留或删除分支后的任何指令。

> 惩罚是固定的。
> 无法通过软件减少。

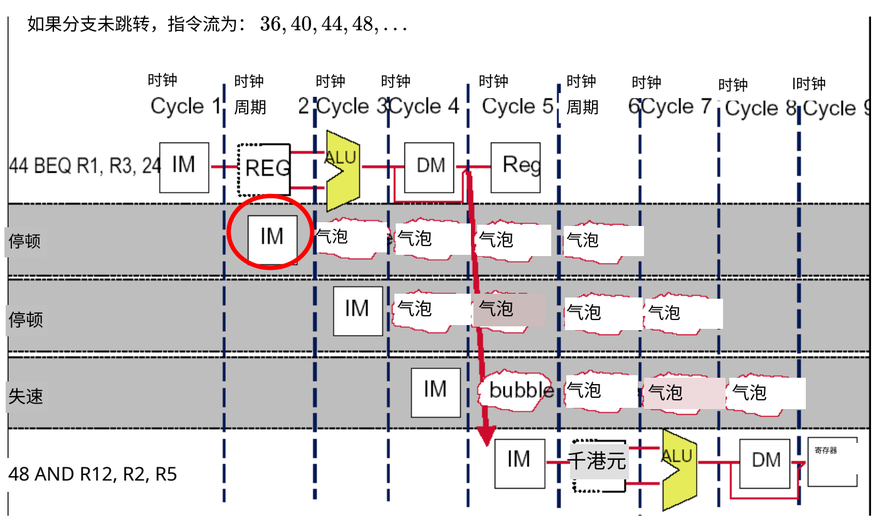
☐ **实现方式。注意与数据冒险停顿的区别**
> 停止后续指令：保持程序计数器（PC）不变
> 空操作 → IF/ID.IR清除错误获取的指令
> 空操作 → ID/EX.IR向前推送一个气泡

停顿会极大地损害性能

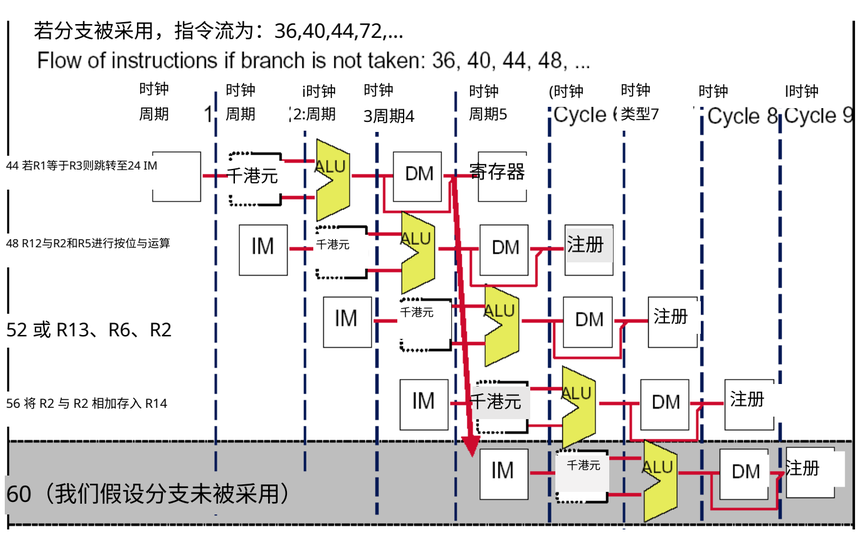
☐ **问题：**
> 在分支频率为30%且理想CPI为1的情况下，插入停顿会对性能产生多大影响?

[答案：
>消费者物价指数（CPI） = 1 + 30% × 3 = 1.9
›•这种简单的解决方案只能达到理想性能的约一半。

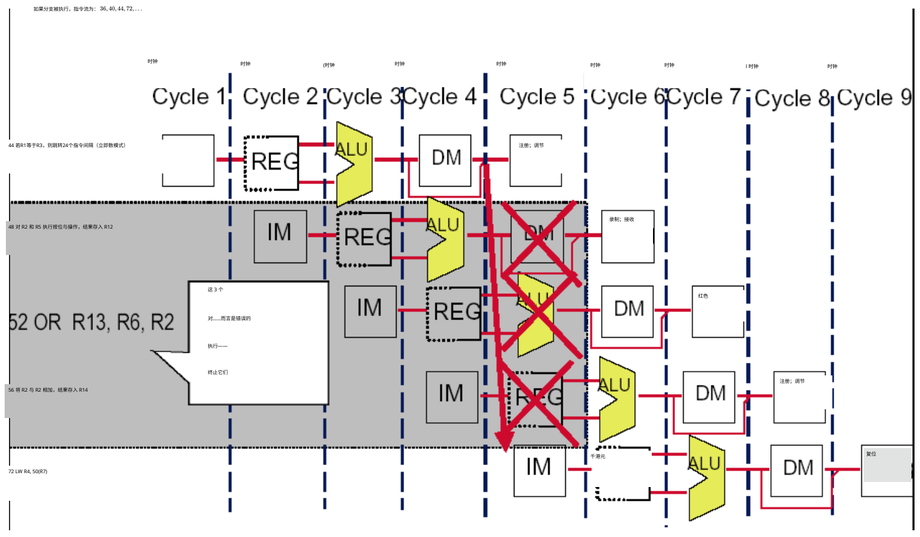
始终停顿会影响未跳转情况



假设分支未跳转怎么样



如何处理已执行的分支?



预测不跳转

☐ **硬件:**

> **将每个分支都视为未被采用（或视为正式指令）**

- 当分支未被采用时，获取的指令会继续执行。完全不会出现停顿。
- 如果分支被采用，则在分支目标处重新开始获取指令，这会导致3个周期的停顿。(应将获取的指令转换为空操作)
- 性能 = 1+ 分支率（未采用率 * 0 + 采用率 * 3）

☐ **编译器:**

> 可以通过在未采用路径中编写最常见的情况来提高性能。

未取预测的实现

当分支未被选取时，将该分支当作算术逻辑单元（ALU）指令来执行。

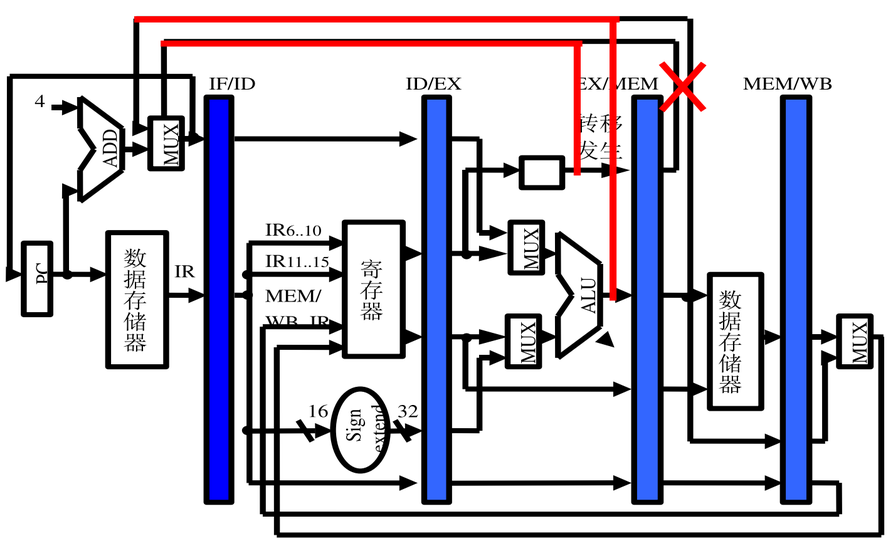
当分支被选取时，则

> 将程序计数器（PC）更改为分支目标地址（转向正确方向）

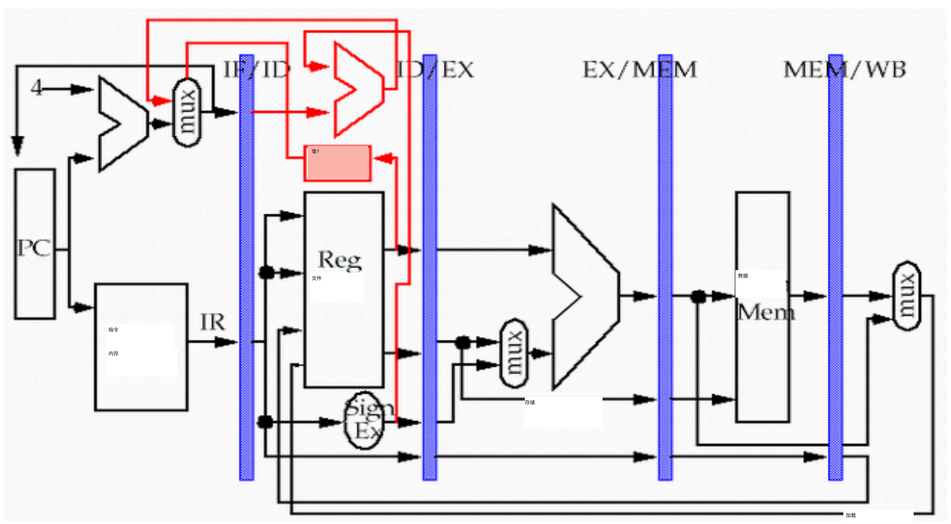
> 终止已被预测要执行的错误指令。

- 空操作 → 取指/译码
- 空操作 → 译码/执行
- 空操作 → 执行/访存

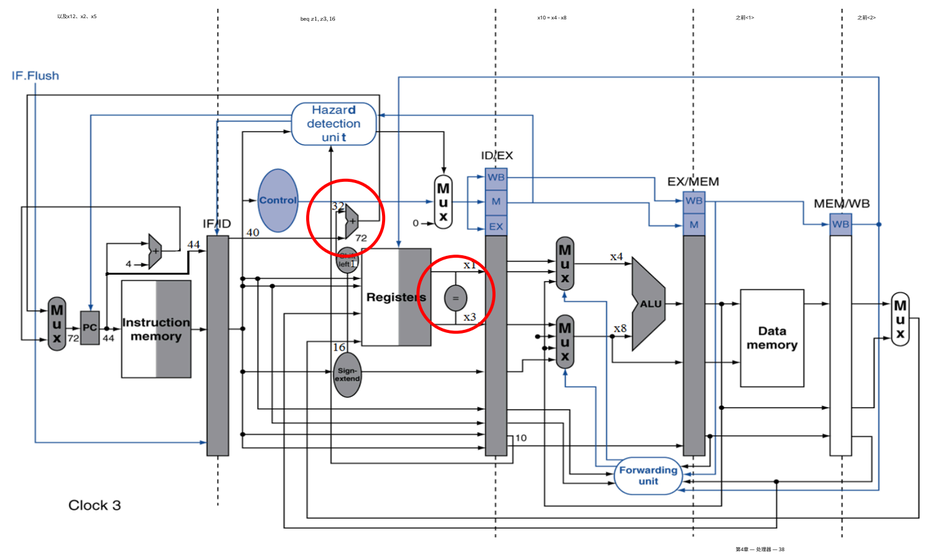
将分支计算提前



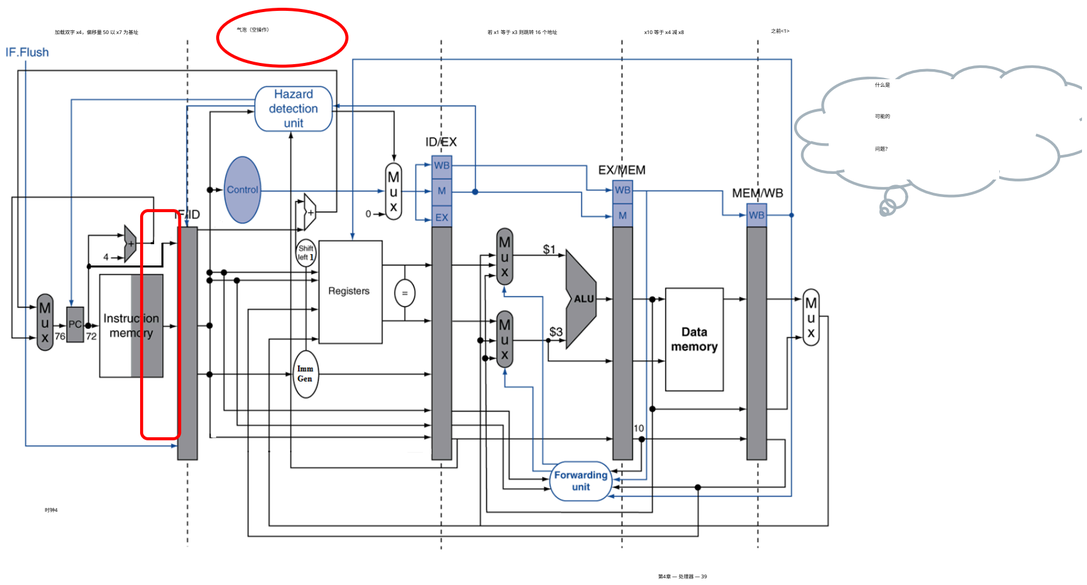
将分支计算提前



示例：分支跳转



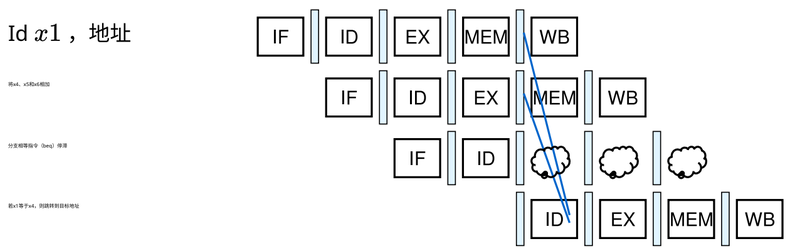
示例：分支已跳转



分支的数据冒险

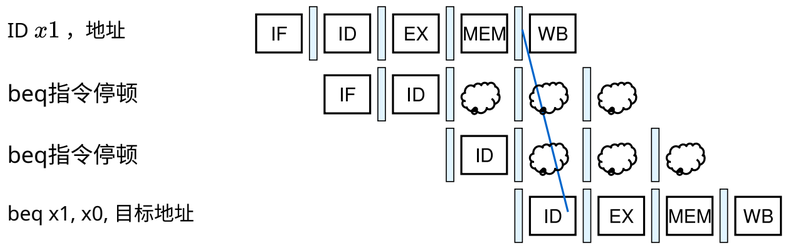
☐如果比较寄存器是前一条ALU指令或2^{nd} 前一条加载指令的目标寄存器

> 需要1个停顿周期



分支的数据冒险

□如果比较寄存器是紧邻的加载指令的目标寄存器 > 需要2个停顿周期

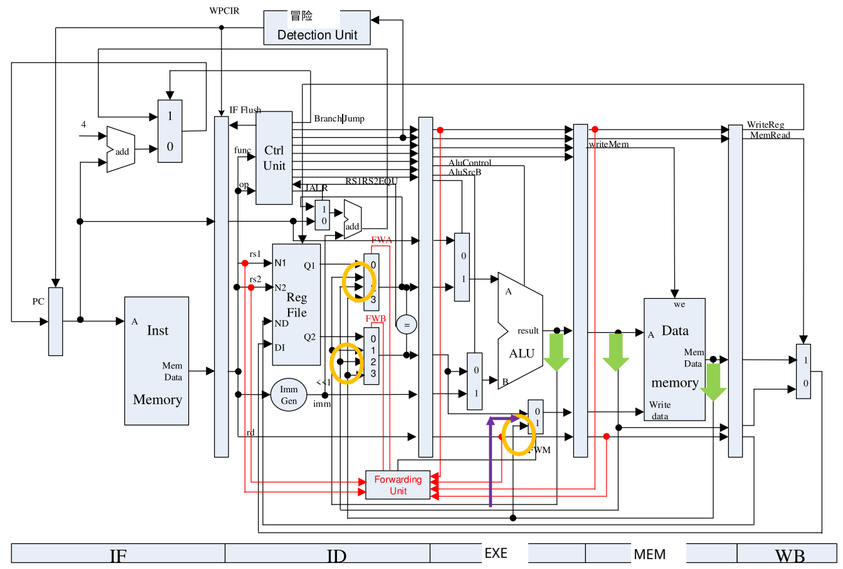


上一页的停顿能否减少？

[是的。移动
转发路径
从执行（EX）阶段到
指令译码（ID）阶段。

尽管代价是
时钟周期时间会
增加。

请注意：
转发路径的源/汇的位置
源/汇
转发的
路径。



问题：何时需要紫色转发路径？

实验1中的前递路径