RISC-V 流水线处理器

本讲内容

- **单周期处理器的性能分析**
- **五阶段流水线处理器、控制信号**
- 指令流水线中的相关性、冒险
  - 结构相关性
  - 数据相关性
  - 控制相关性

- 本讲对应教材章节：
  - 计算机组成与设计，第四章 4.5 ~ 4.7

单周期处理器：不太被采用， why?

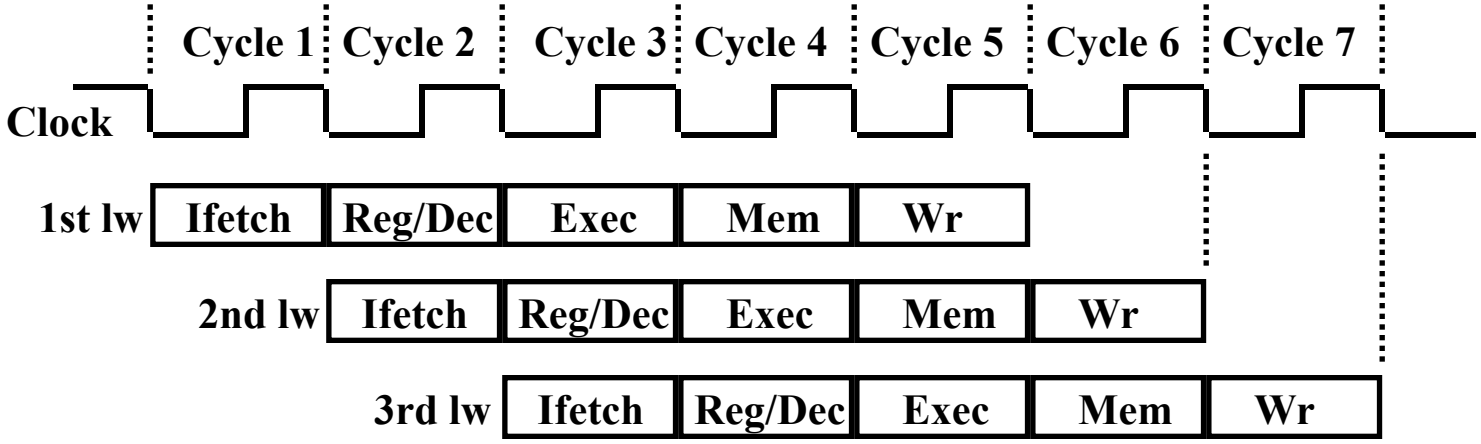
假定主要组成部件的工作时间如下：

- 存储器: 200 ps； ALU 和加法器: 200 ps； 寄存器文件 (读 或 写): 100 ps
- 多路选择器、控制器、PC访问、符号位扩展没有延迟、连接线路也无延迟
- 不考虑寄存器建立时间和锁存延迟、不考虑时钟偏斜

Instr	IF = 200ps	ID = 100ps	ALU = 200ps	MEM=200ps	WB = 100ps	Total
add	X	X	X		X	600ps
beq	X	X	X			500ps
jal	X	X	X			500ps
lw	X	X	X	X	X	800ps
sw	X	X	X	X		700ps

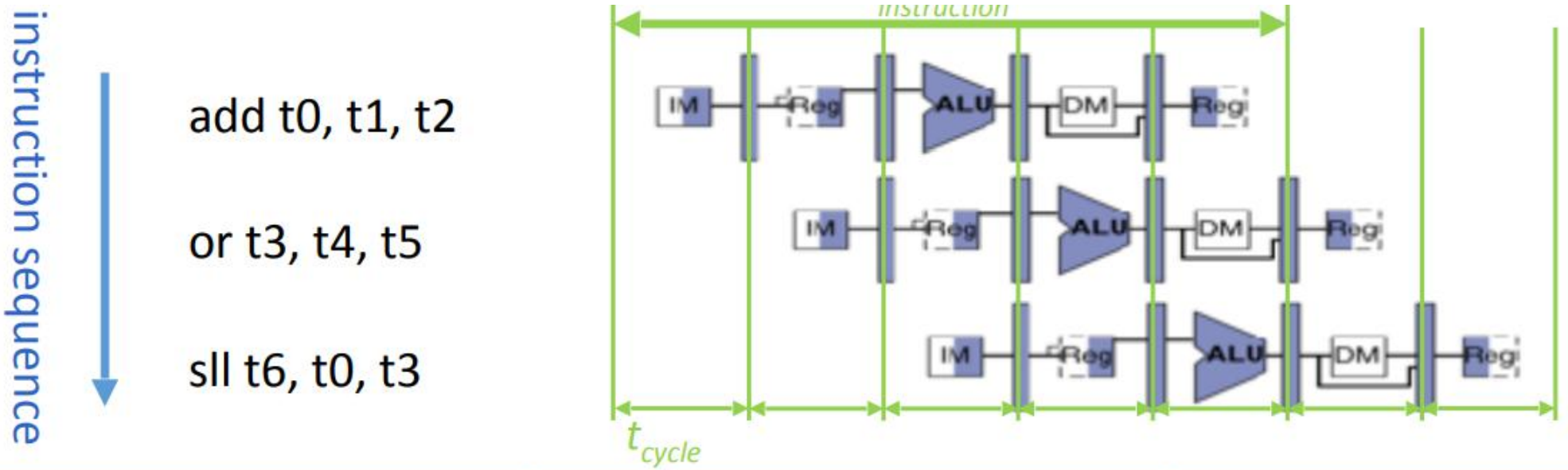
最高的时钟频率= 1/800ps = 1.25 GHz
很多模块大部分时间是空闲的。**How can we keep ALU busy all the time ?**

指令的执行流水化



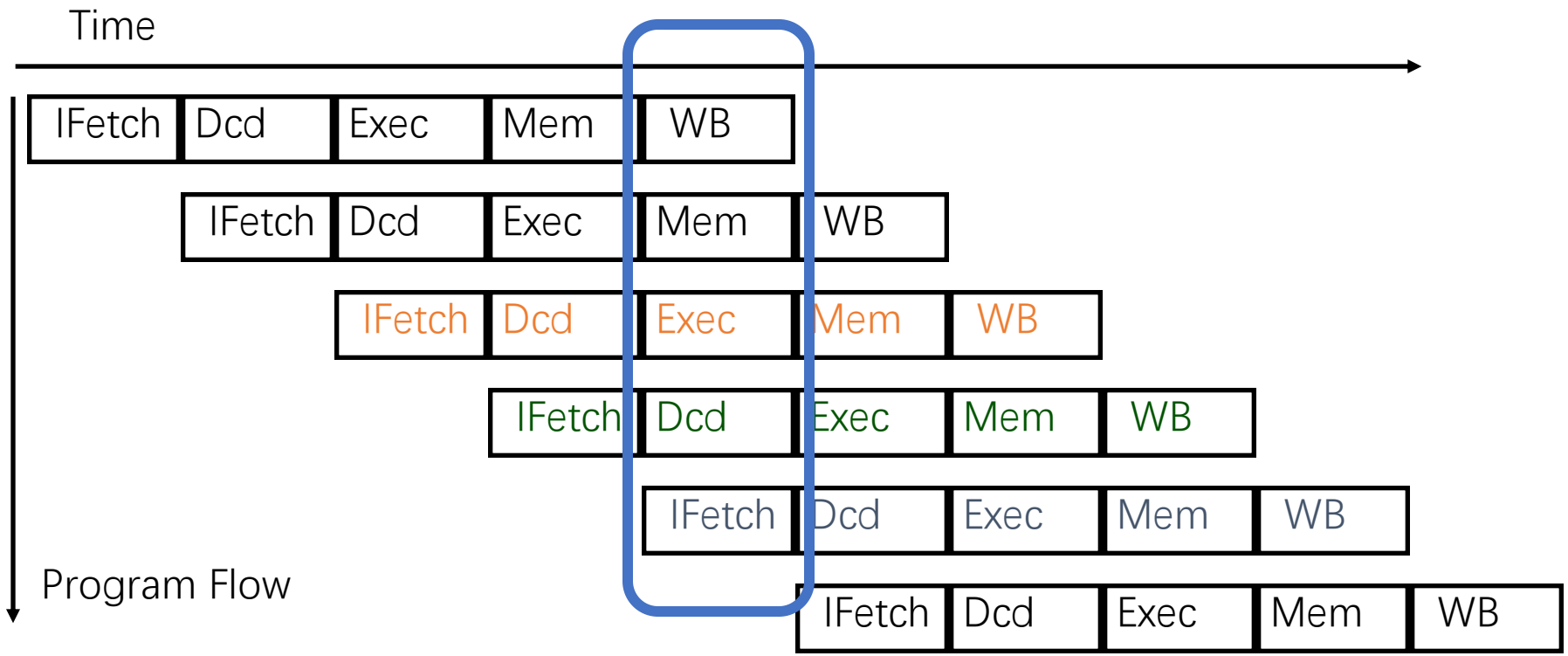
- 在数据通路上有五个功能单元:
  - 指令存储器: Instruction Memory： Ifetch （取指令）阶段
  - 寄存器文件的读端口 (bus A and busB)： Reg/Dec （读寄存器/译码）阶段
  - 算术逻辑运算单元 ALU: Exec（执行）阶段
  - 数据存储器 Data Memory： Mem（访存）阶段
  - 寄存器文件的写端口(bus W)： Wr（写结果） 阶段

RISC-V指令流水线



	Single Cycle	Pipelining
Timing	t_{step} = 100 … 200 ps	t_{cycle} = 200 ps
	Register access only 100 ps	All cycles same length
Instruction time, $t_{instruction}$	= t_{cycle} = 800 ps	1000 ps
Clock rate, f_s	1/800 ps = 1.25 GHz	1/200 ps = 5 GHz
Relative speed	1 x	4 x

理想的指令流水线

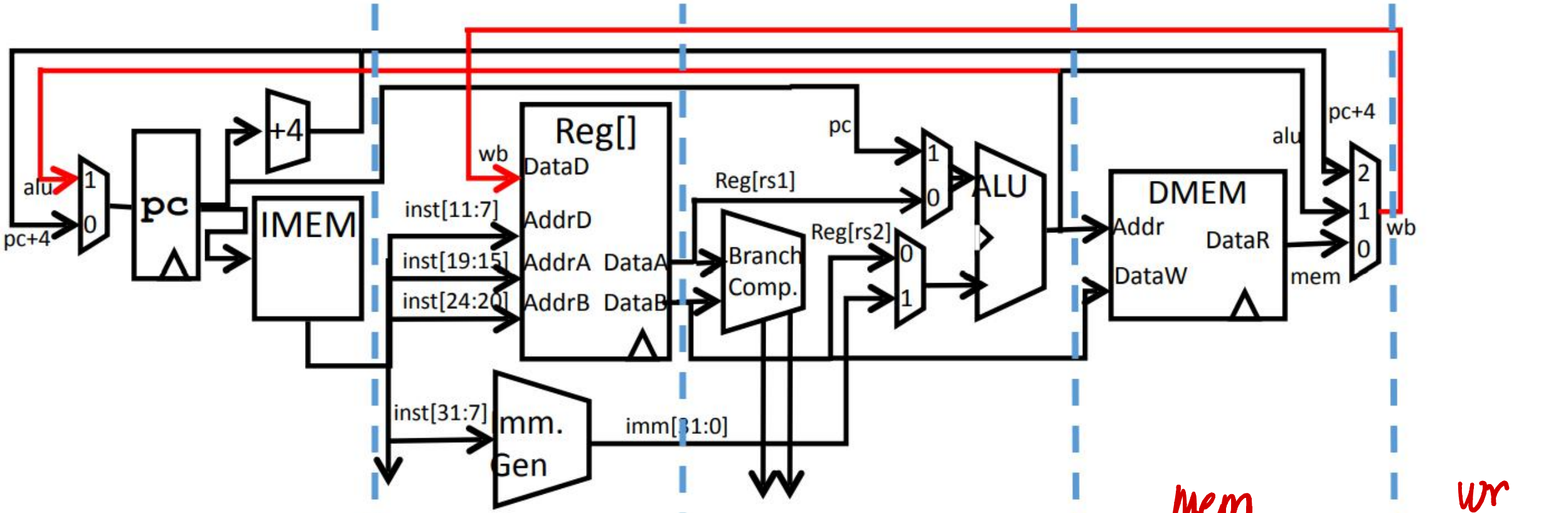


理想情况下，每一个周期
完成一条老的指令；
开始一条新的指令。达到CPI=1；
CPI：完成一条指令所花的周期数

cycle per instruction

指令流水线的实现

Pipelining RISC-V RV32I Datapath



Instruction Fetch (F) IF

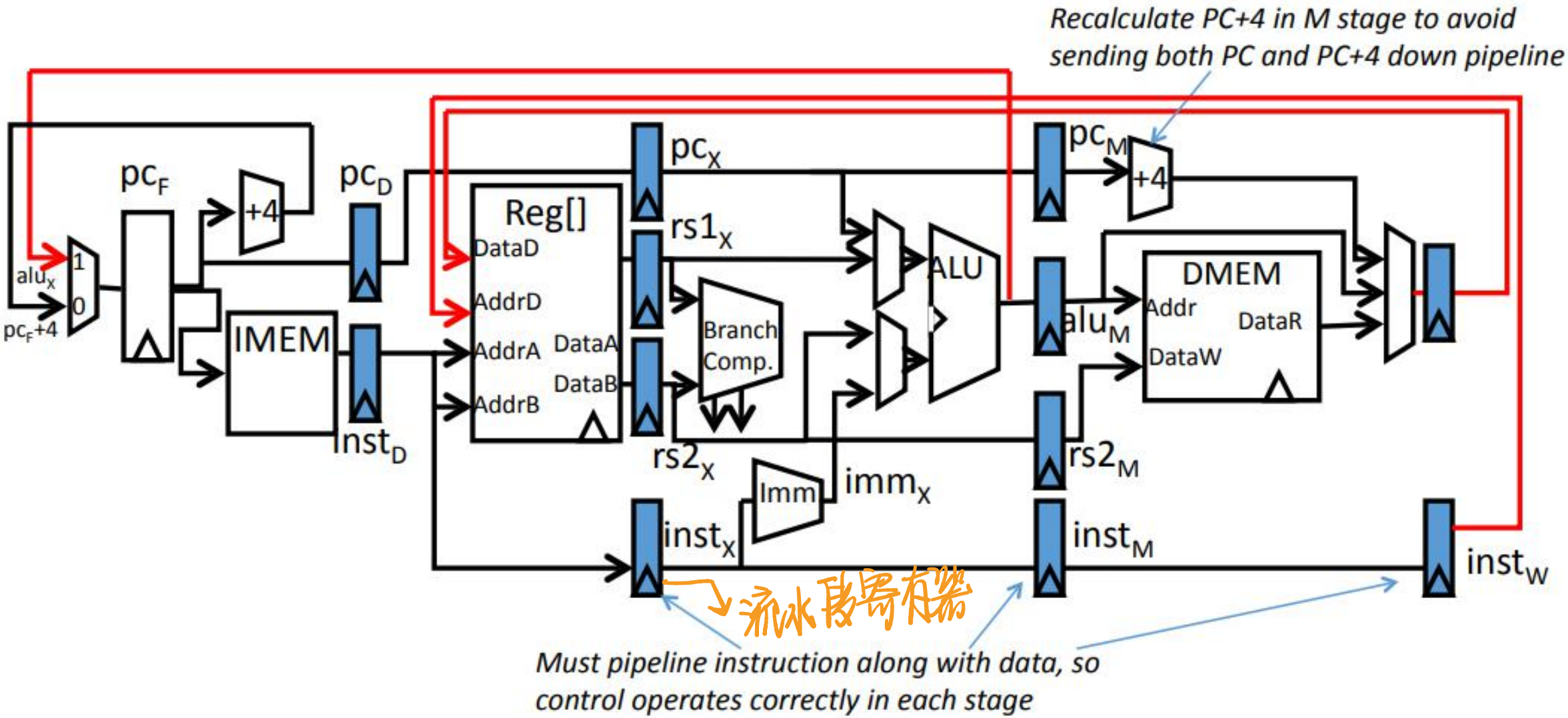
Instruction Decode/ Register Read (D) Reg

ALU Execute (X) exe

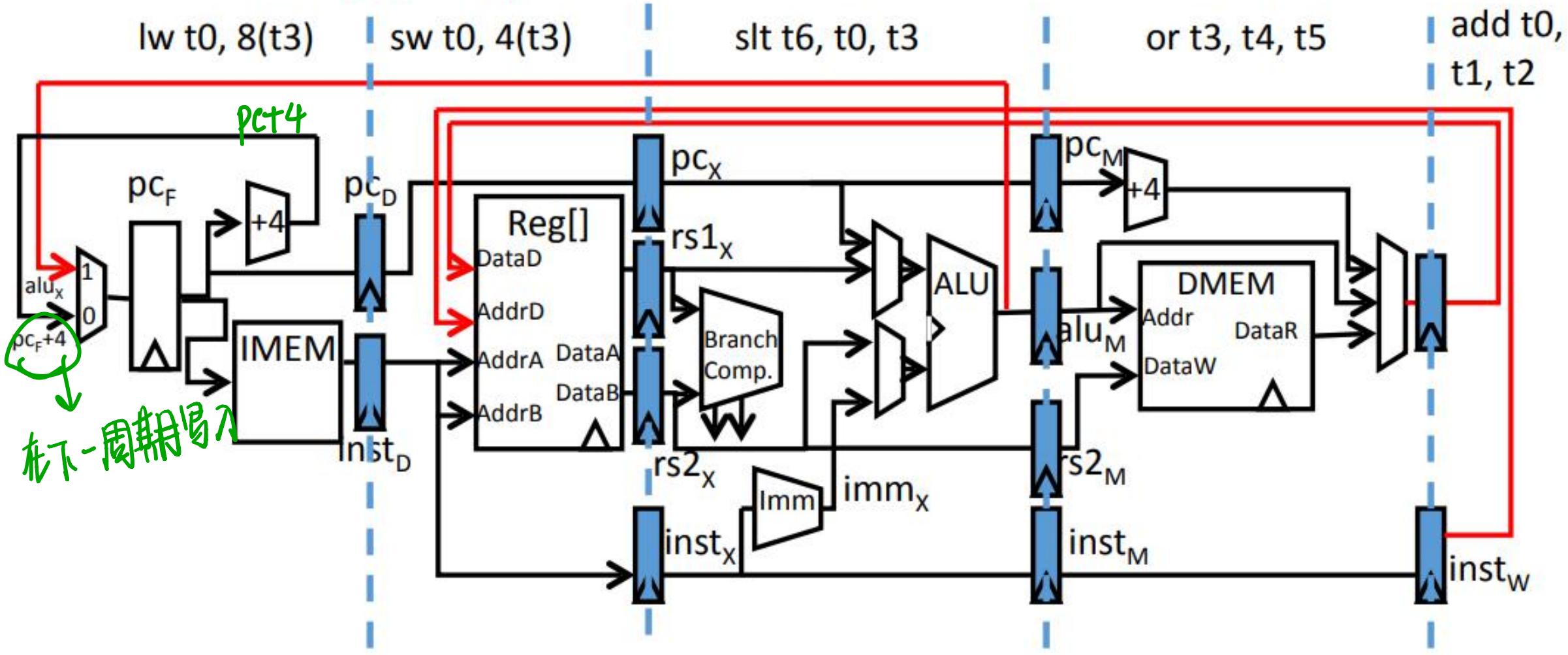
Memory Access (M) mem

Write Back (W) wr

Pipelined RISC-V RV32I Datapath



Each stage operates on different instruction

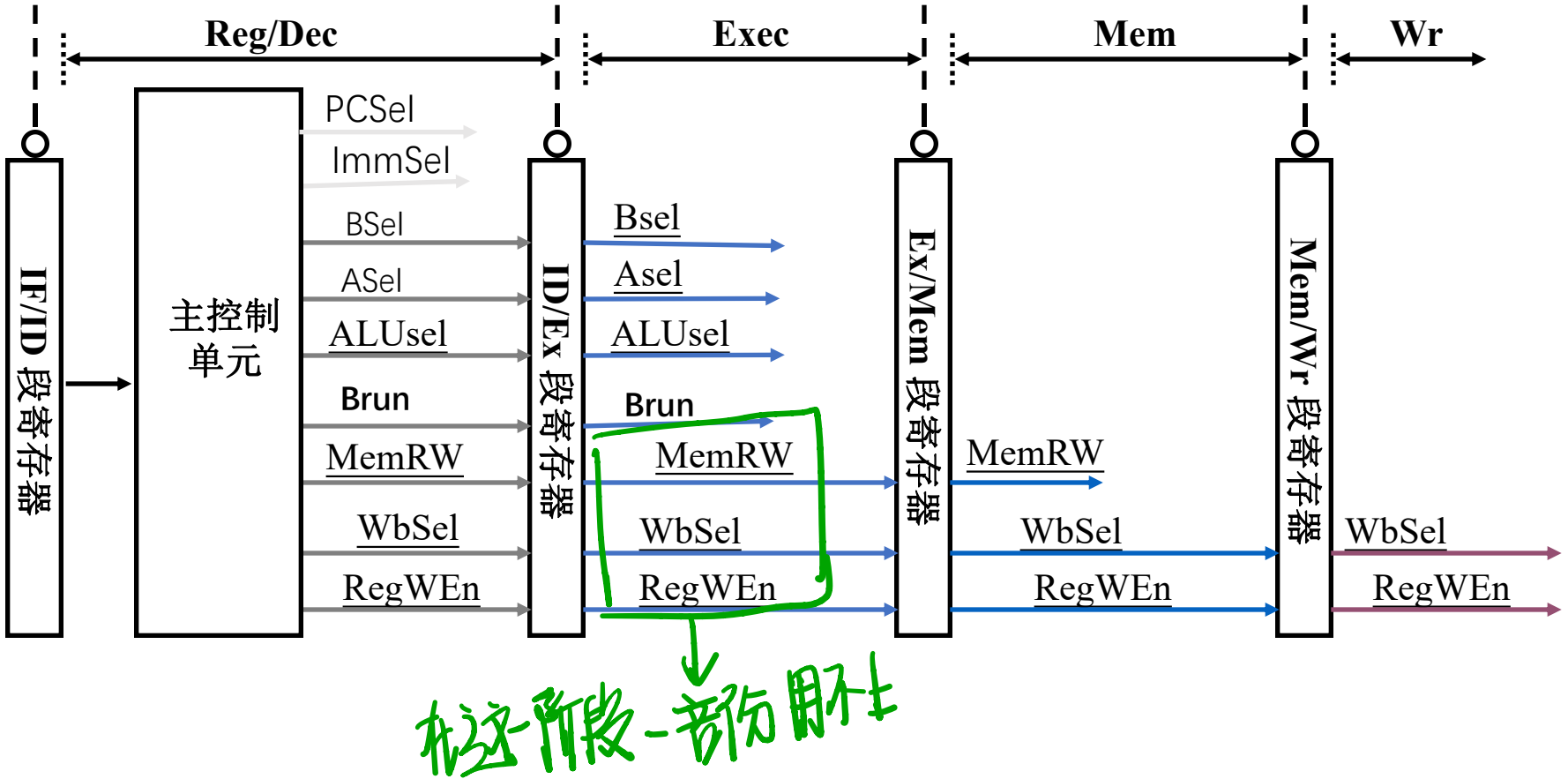


Pipeline registers separate stages, hold data for each instruction in flight

控制信号的传递

- 主控制单元在译码段（ Reg/Dec ）产生所有控制信号
  - Exec 段需要的控制信号，在一周期后使用
  - Mem 段需要的控制信号，在两周期后使用
  - Wr 段需要的控制信号，在三周期后使用

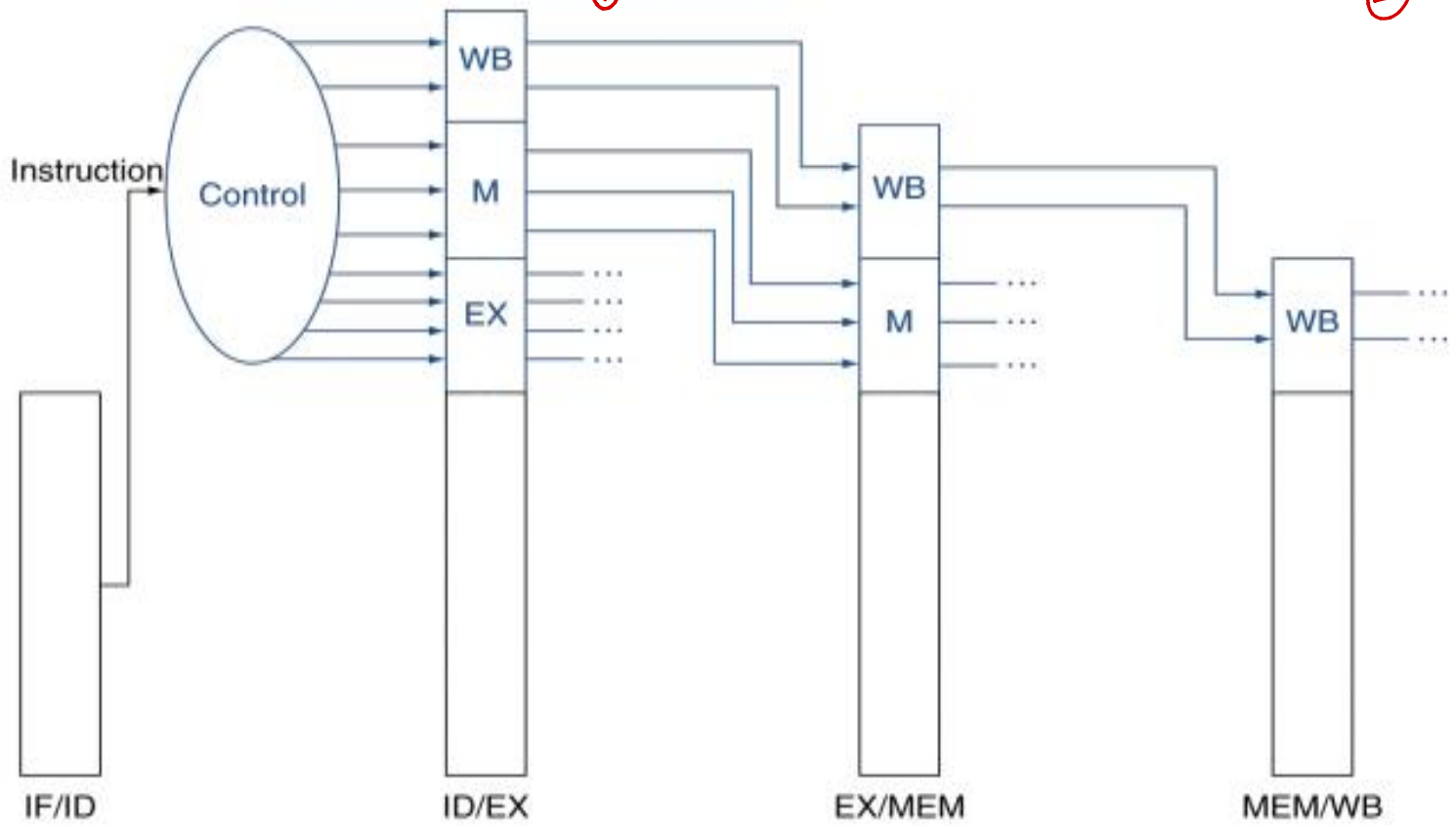
每个阶段寄存器数可以不同.



在这阶段-无作用

流水线处理器的控制信号

- 控制器的实现和单周期实现的控制器一样
- 控制信号存储在段寄存器、沿着流水段逐级传递
- 段寄存器除了存储上一段的结果，还存储本段以及后面段所用的控制信号



问题：关于流水线处理器，哪一个是错误的？

A. 流水线方式下，流水段寄存器增加了延迟，一条指令的执行时间反而加长了。但由于流水线执行提升了指令吞吐率，程序总的执行时间缩短了；

B. 指令规整、指令长度固定、指令条数少的指令系统更易于实现指令流水线；

C. 流水段之间的流水段寄存器(stage register) 用于传递指令执行的中间结果和控制信号，由于各个指令执行阶段的中间结果和控制信号不一样，各流水段寄存器的宽度不一样。

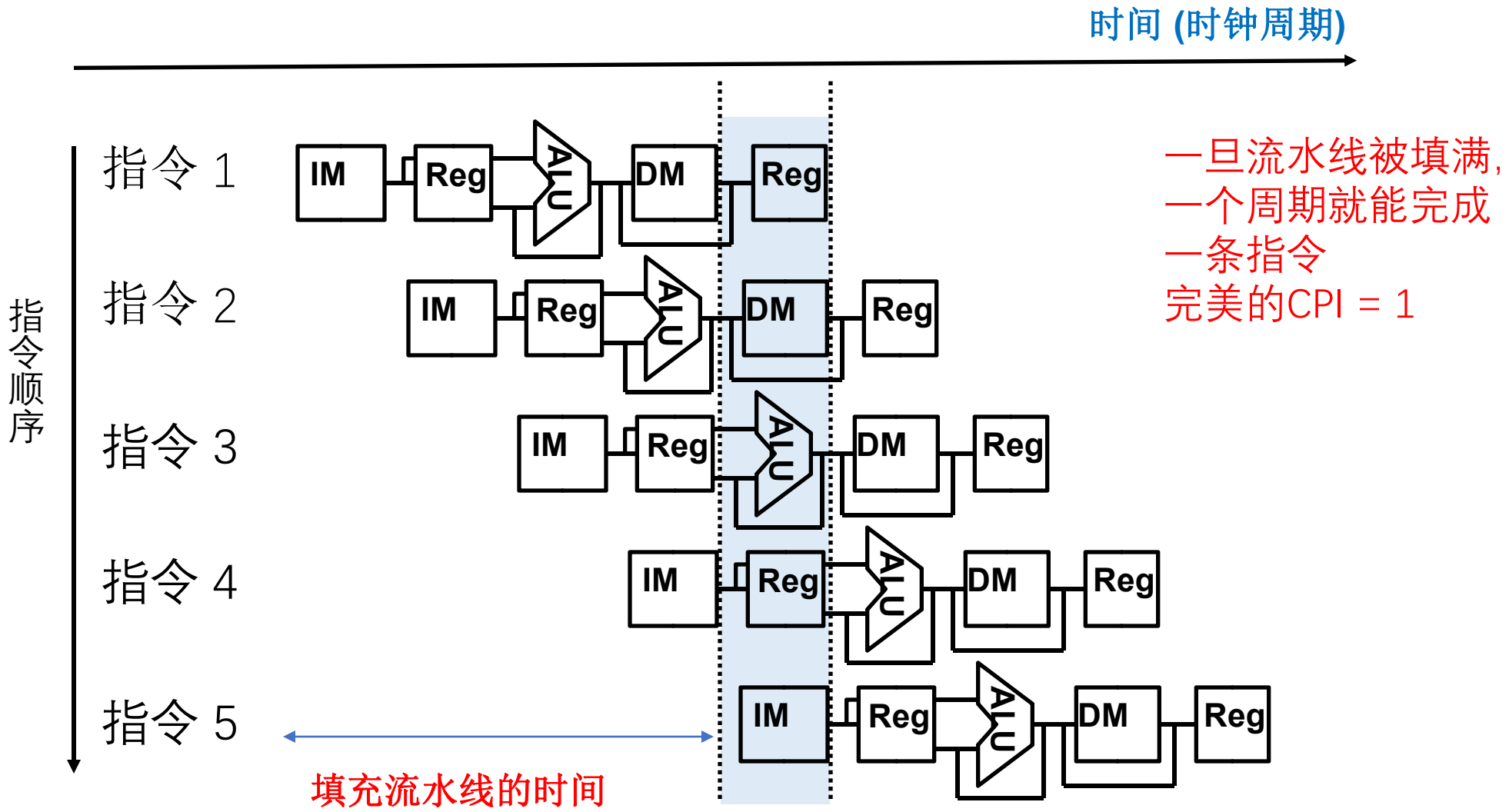
D. 流水线处理器中时钟周期根据最长的流水段延迟来确定。在流水执行方式下，每个时钟周期一定会有一条指令完成，单条流水线处理器的CPI=1

本讲内容

- 单周期处理器的性能分析
- 五阶段流水线处理器、控制信号
- **指令流水线中的相关性、冒险**
  - 结构相关性
  - 数据相关性
  - 控制相关性

- 本讲对应教材章节：
  - 计算机组成与设计，第四章 4.5 ~ 4.7

流水线可以提高吞吐率



一旦流水线被填满，
一个周期就能完成
一条指令
完美的CPI = 1

流水线中的相关性和“冒险”

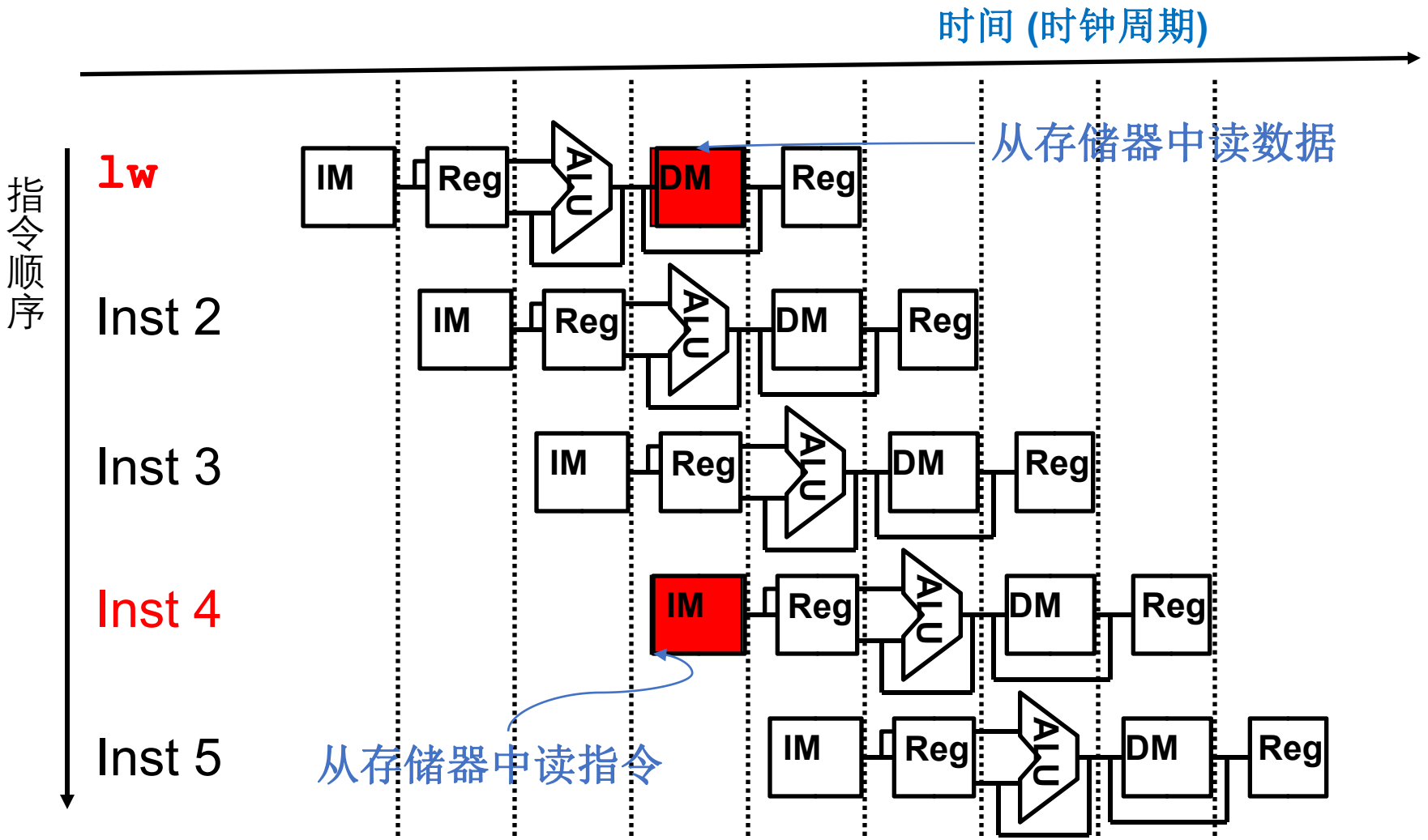
- 流水线中的相关性：相邻或相近的指令之间因存在某种依赖性，或称为相关性，使得指令的执行可能受到影响。
- 这些相关性，可能会影响指令的执行，也可能不影响，因此又称为冒险（hazard）

- 结构冒险 （ structural hazards ）
  - 资源相关性：所需的硬件部件正在为之前的指令工作
- 数据冒险（ data hazards ）
  - 数据依赖性需要等待之前的指令完成数据的读写
- 控制冒险（ control hazards ）
  - 转移指令引起：需要根据指令的结果决定下一步

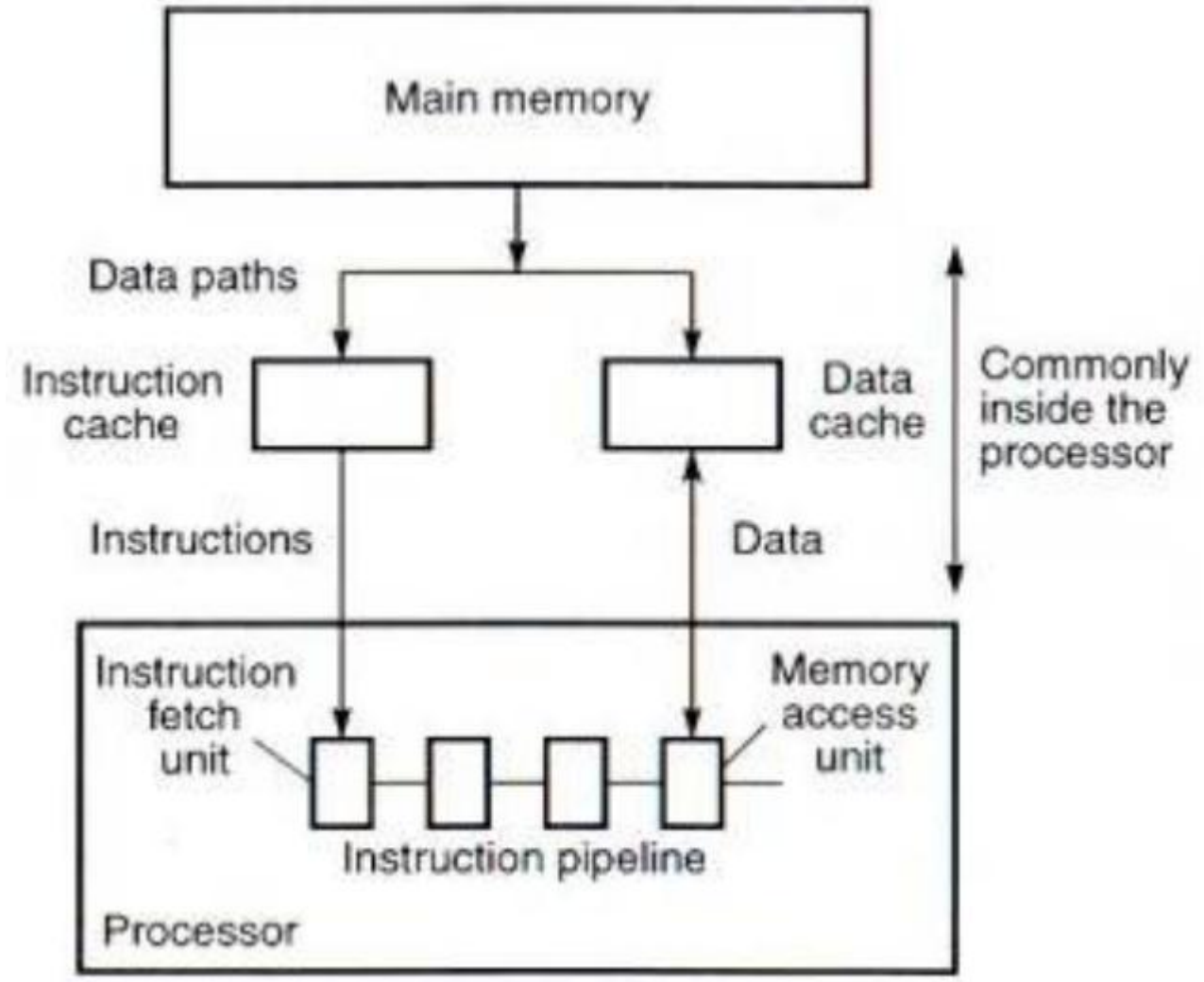
结构冒险

- 结构冒险 （structural hazards）
  - 资源相关性：所需的硬件部件正在为之前的指令工作

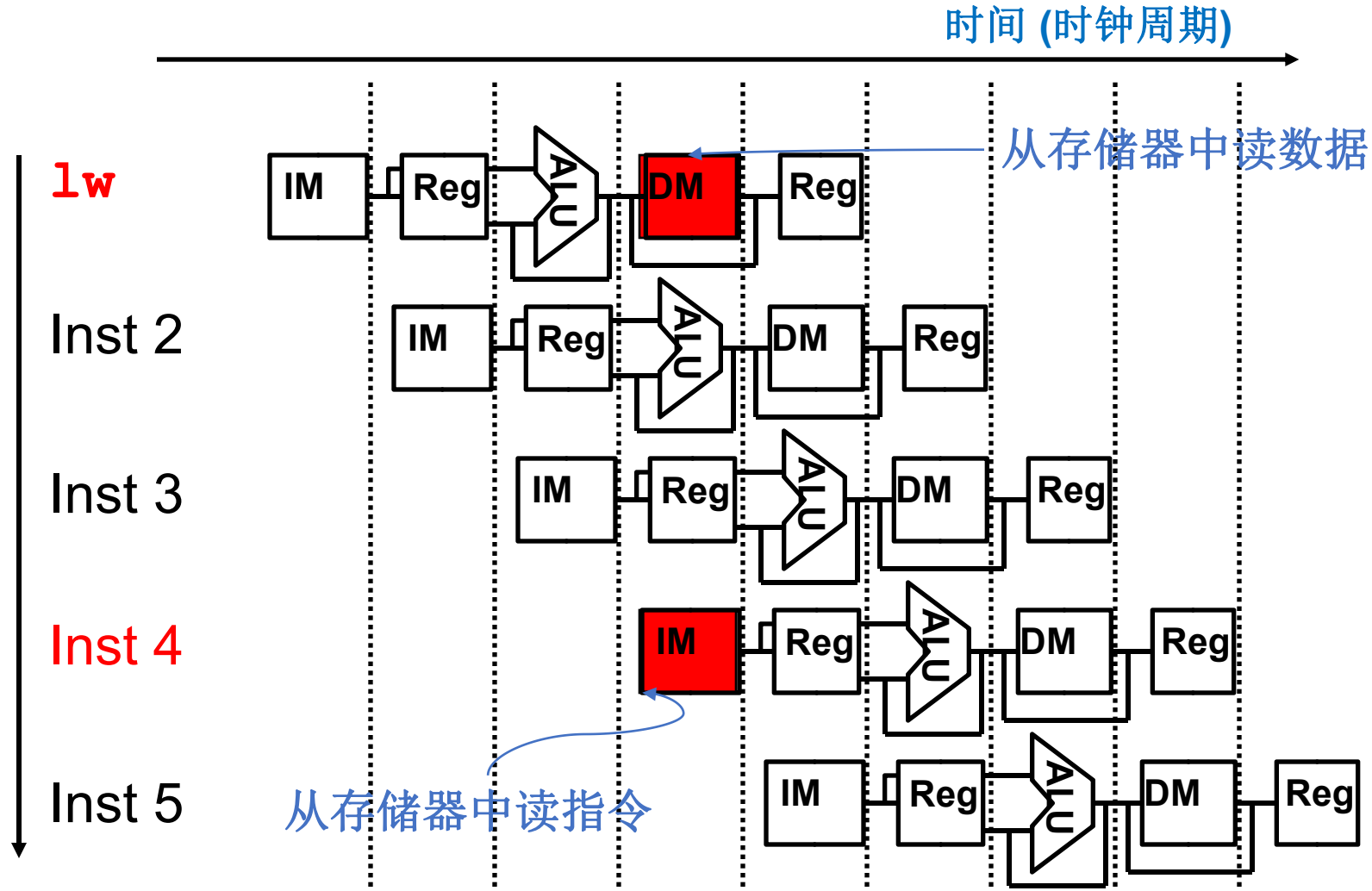
结构冒险：如果只有一个存储器



Solving Structural Hazard with Caches

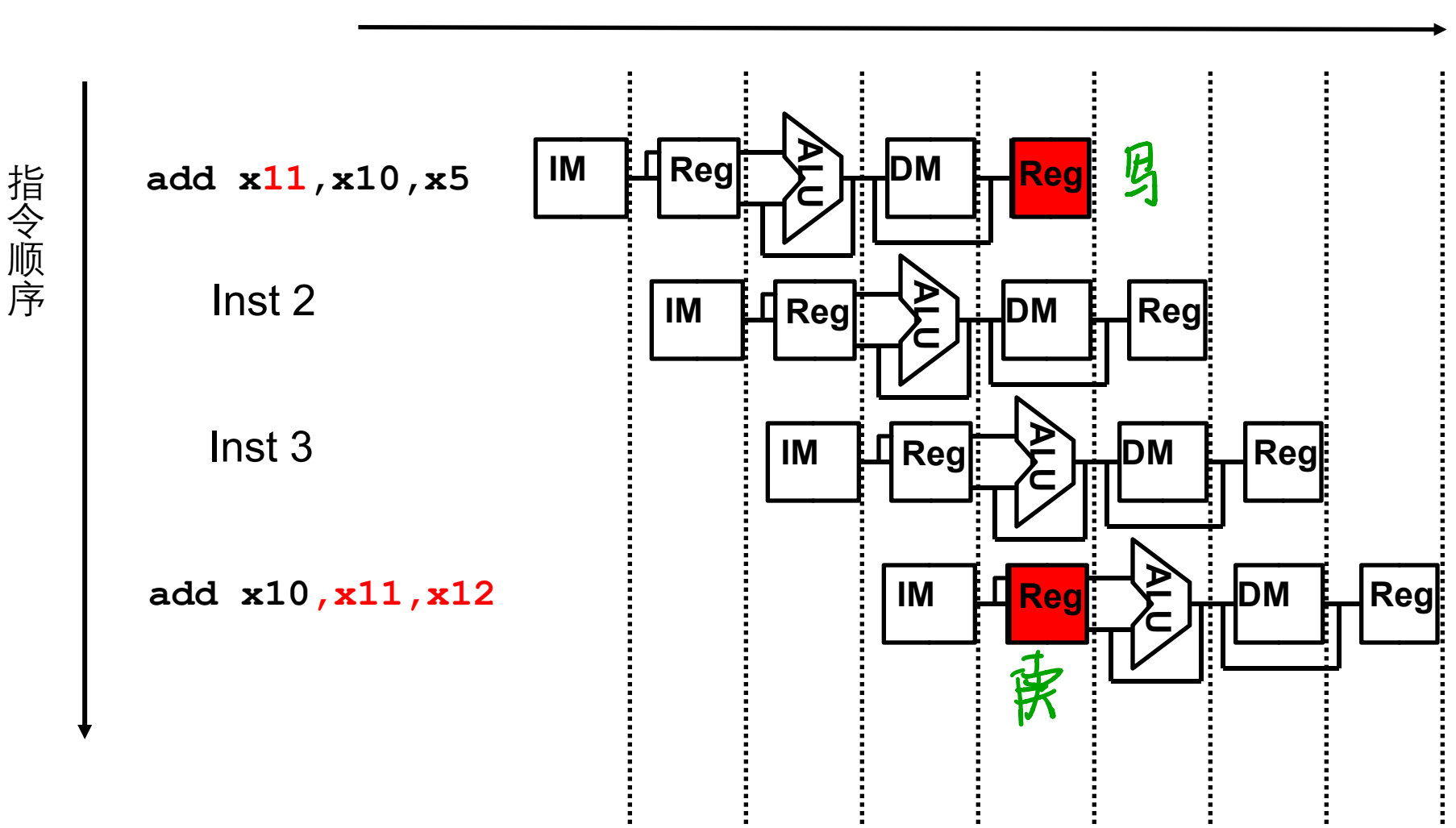


方案：指令存储器和数据存储器相互独立



- 例如：CPU中分离的一级高速缓存：指令cache 数据cache

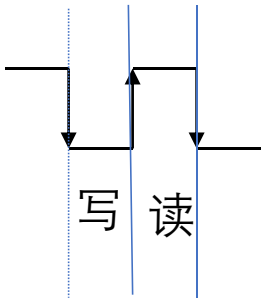
寄存器文件的读写是否存在结构冒险?

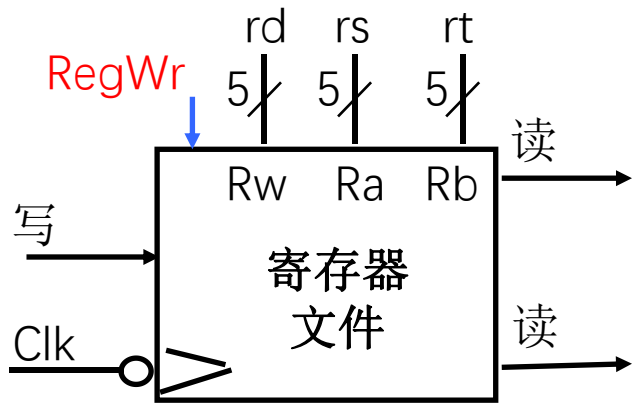


方案：读写口独立的寄存器文件

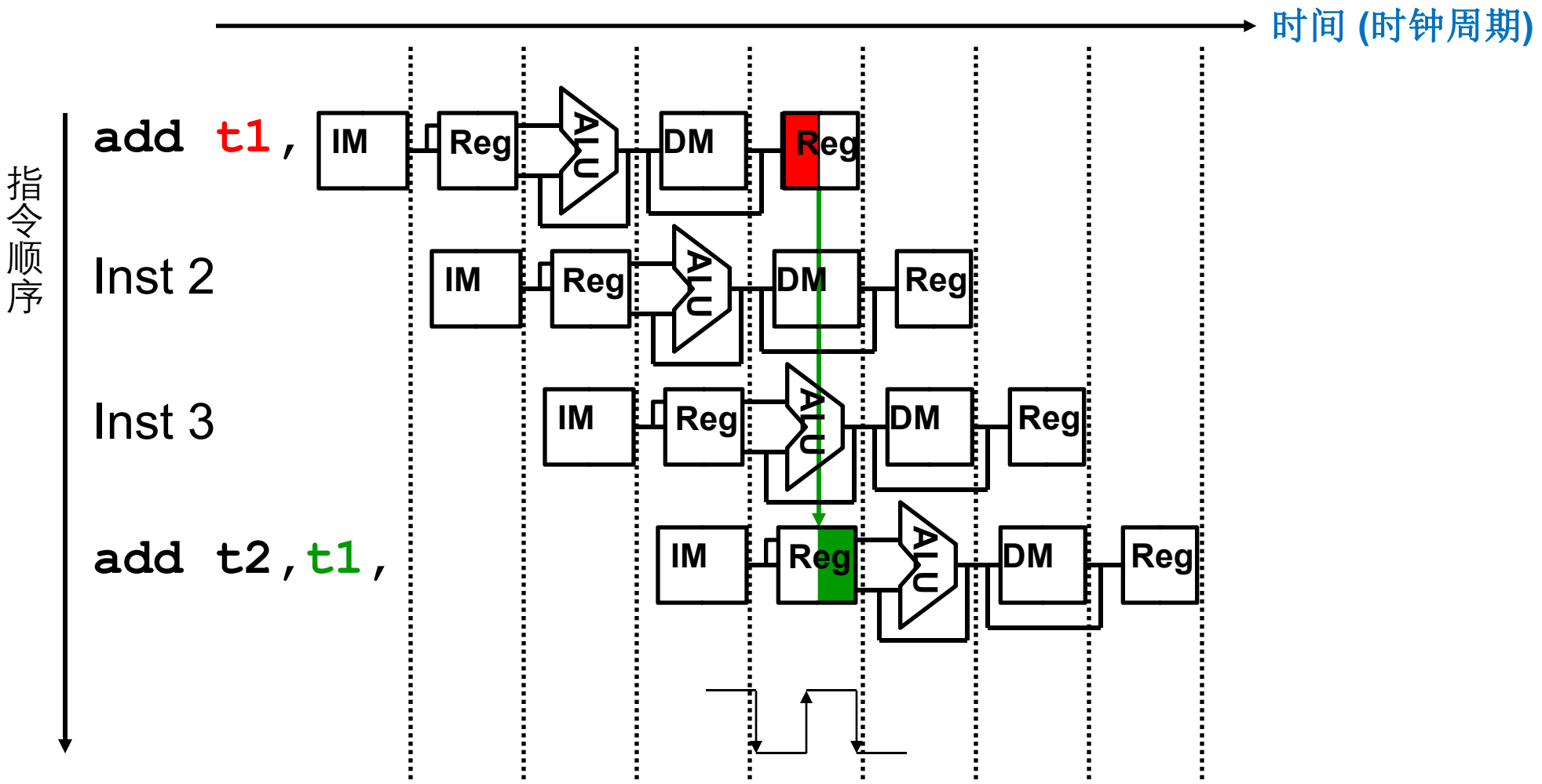
- 寄存器文件的读写速度较快
- 假设
  - 读或写寄存器的延迟为100ps（半个周期）
  - ALU、MEM等部件的延迟为200ps （一个周期）

寄存器文件设置相互独立的读口和写口
写寄存器：在一个周期的前半段完成
读寄存器：在一个周期的后半段完成





寄存器文件的读写不存在结构冒险

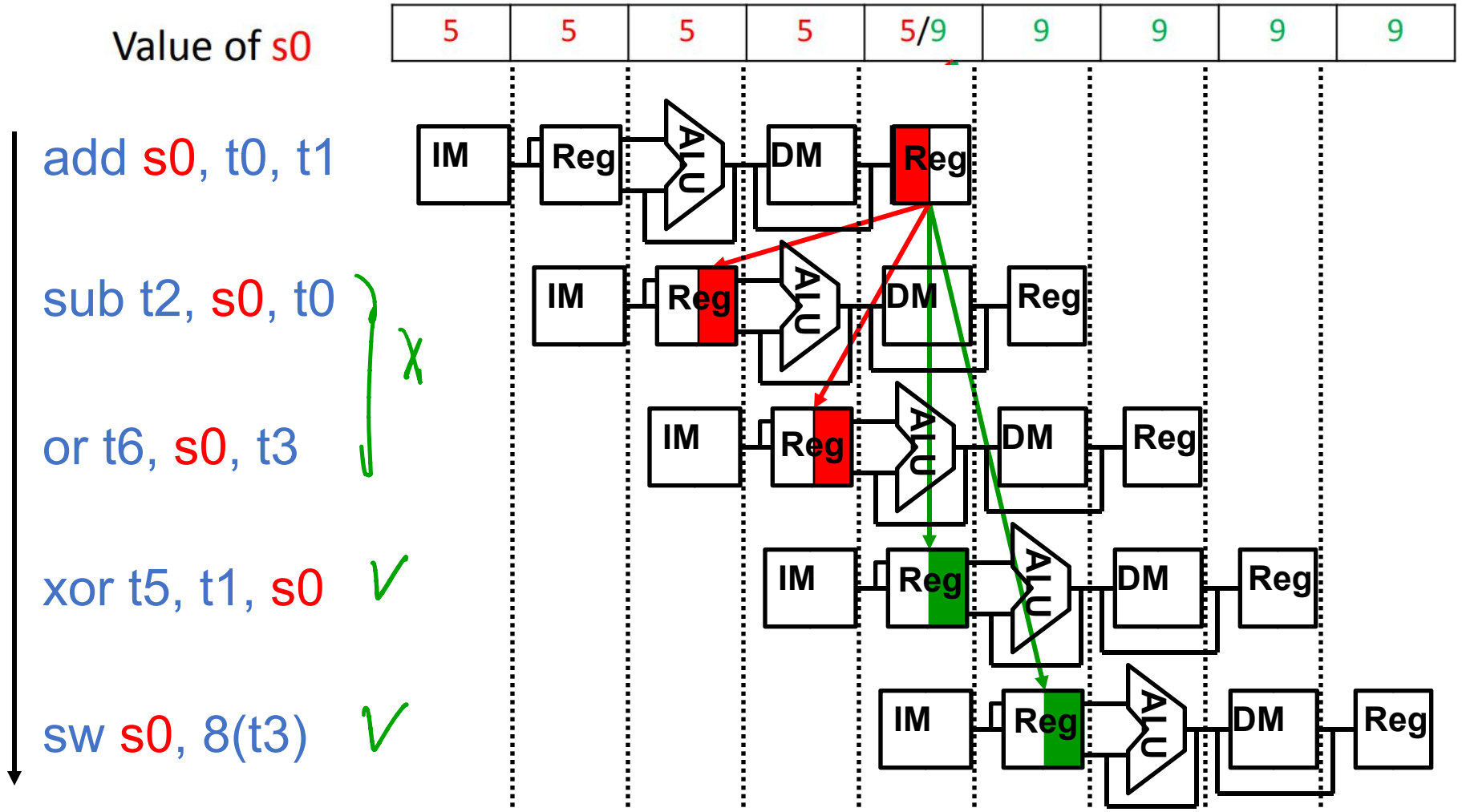


数据冒险

数据冒险（data hazards）

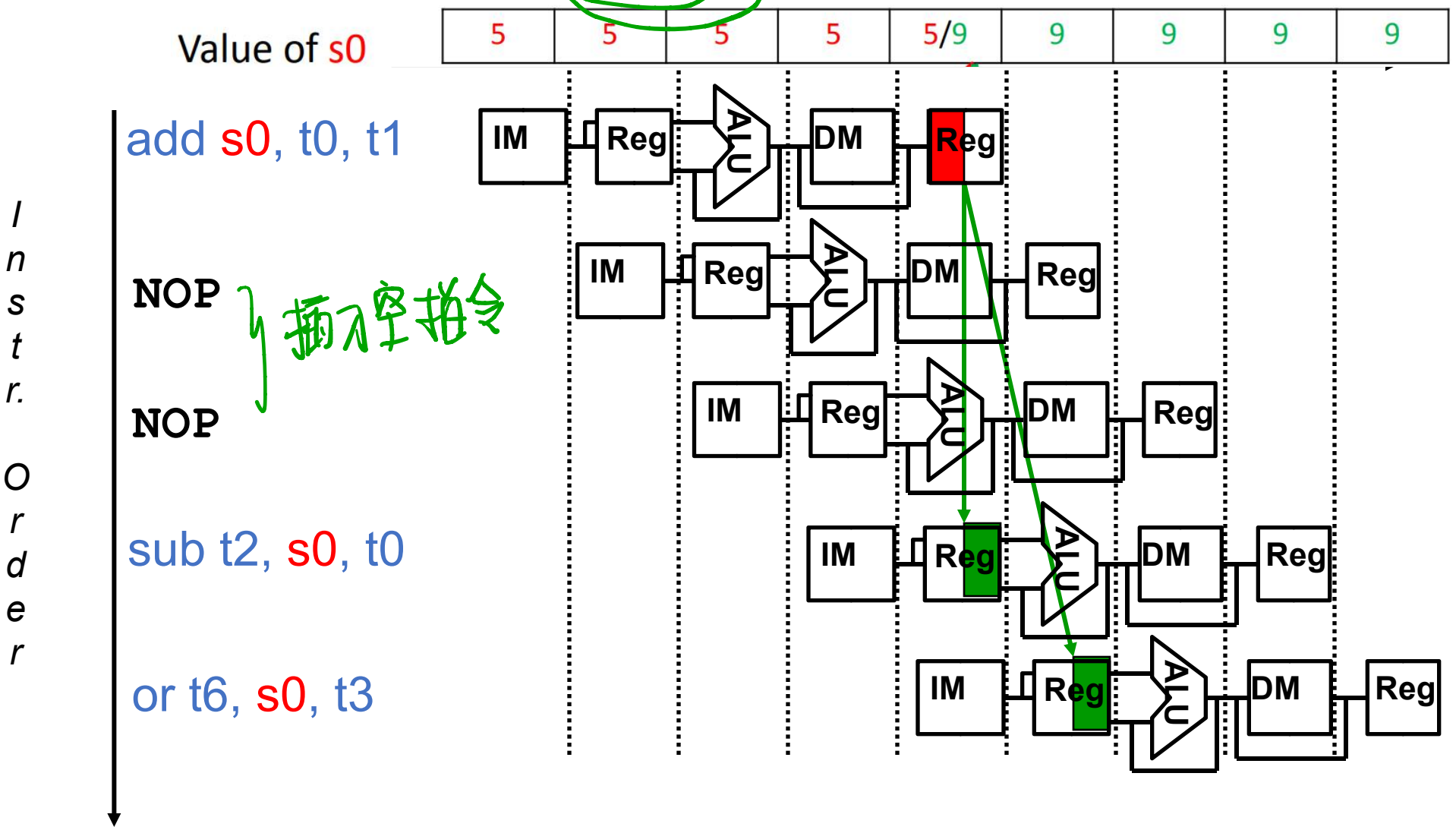
- 数据冒险定义：当一条指令需要等待之前的指令完成数据的读写，不能执行时，就发生了数据冒险
  - R-type
  - Load指令

R-type指令导致的数据冒险

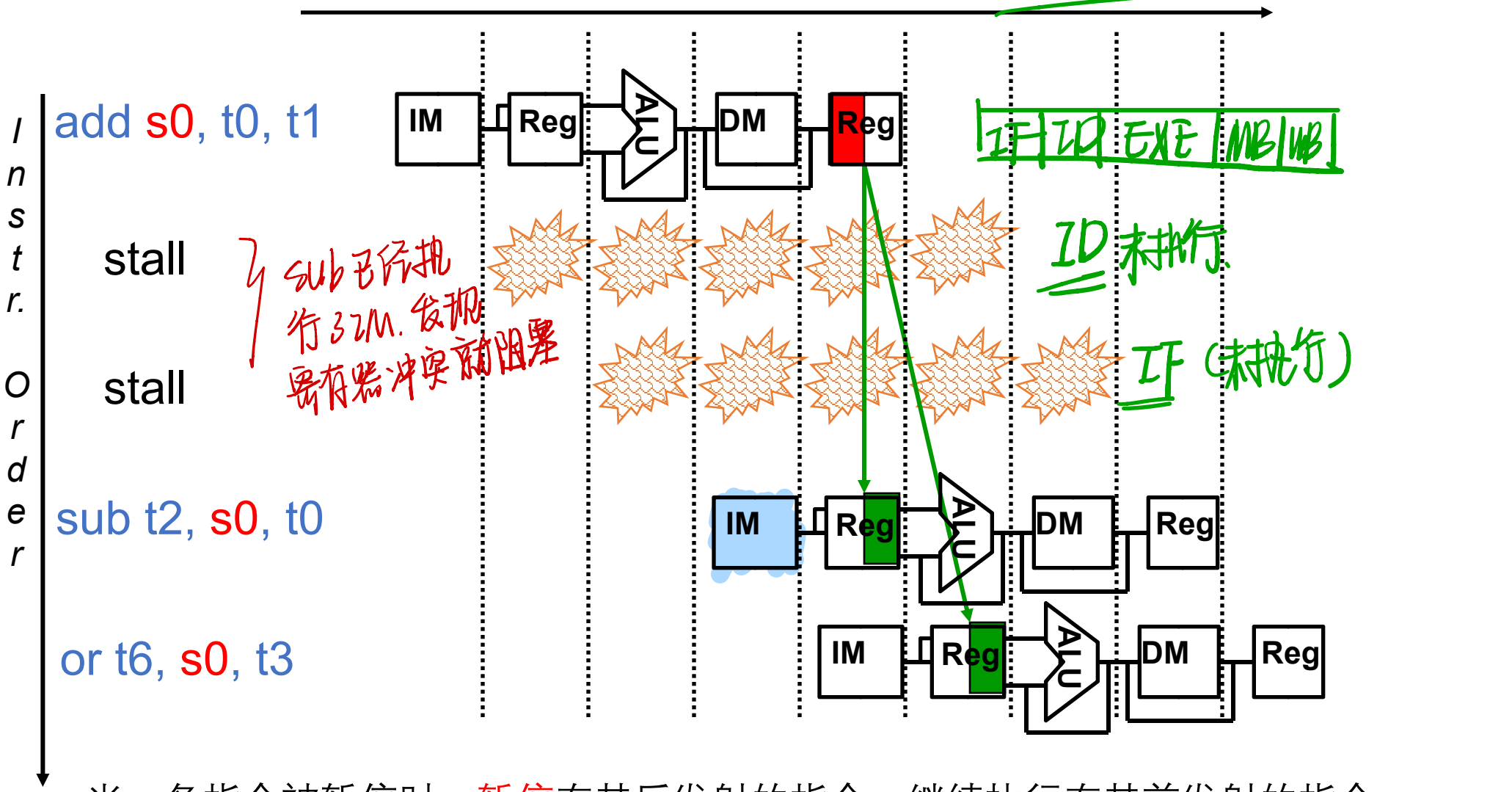


- 写后读数据冒险（Read after write data hazard）

一种简单的软件解决方案：插入空指令

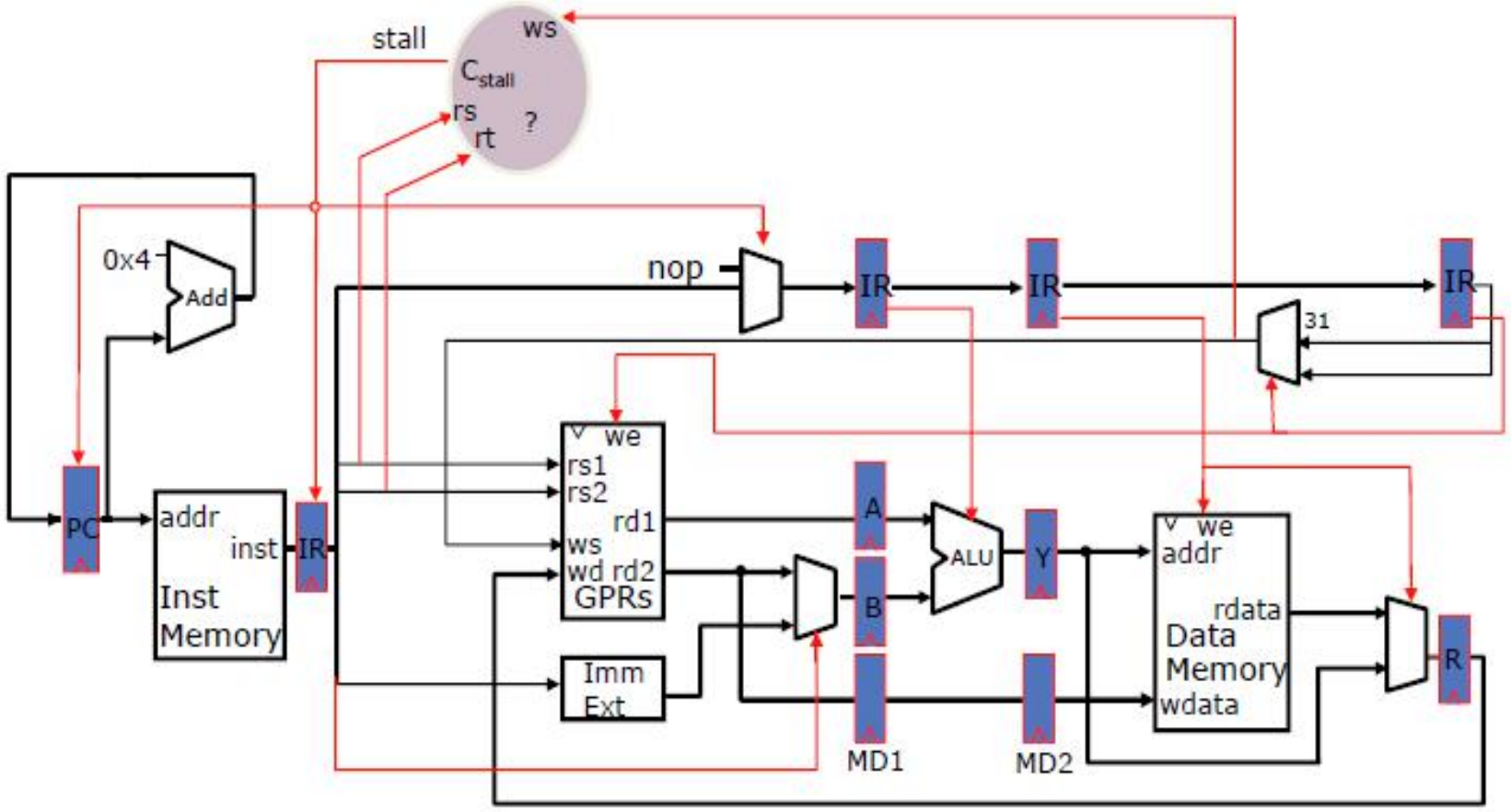


一种简单的硬件解决方案：流水线停顿



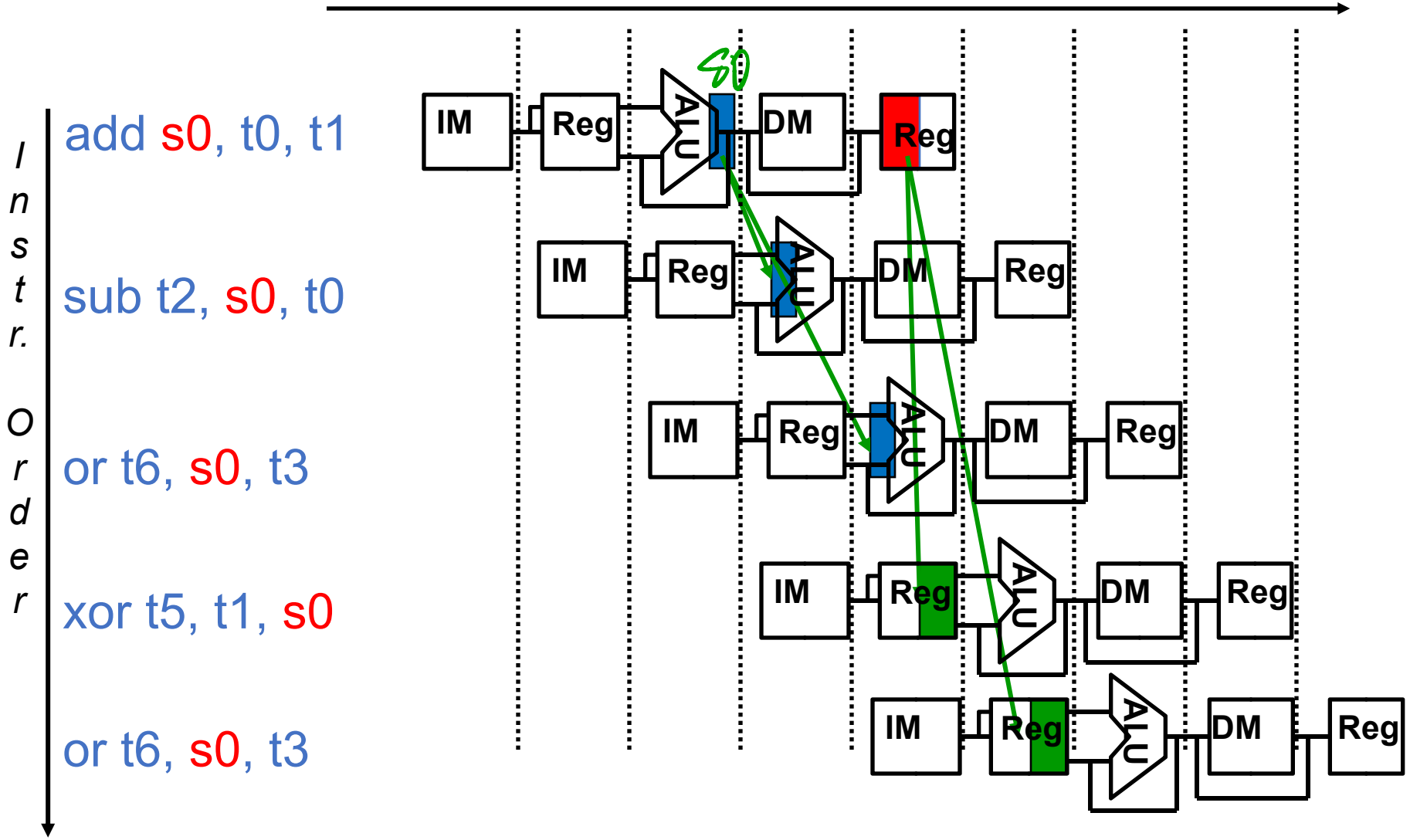
当一条指令被暂停时，暂停在其后发射的指令，继续执行在其前发射的指令。

停顿检测电路



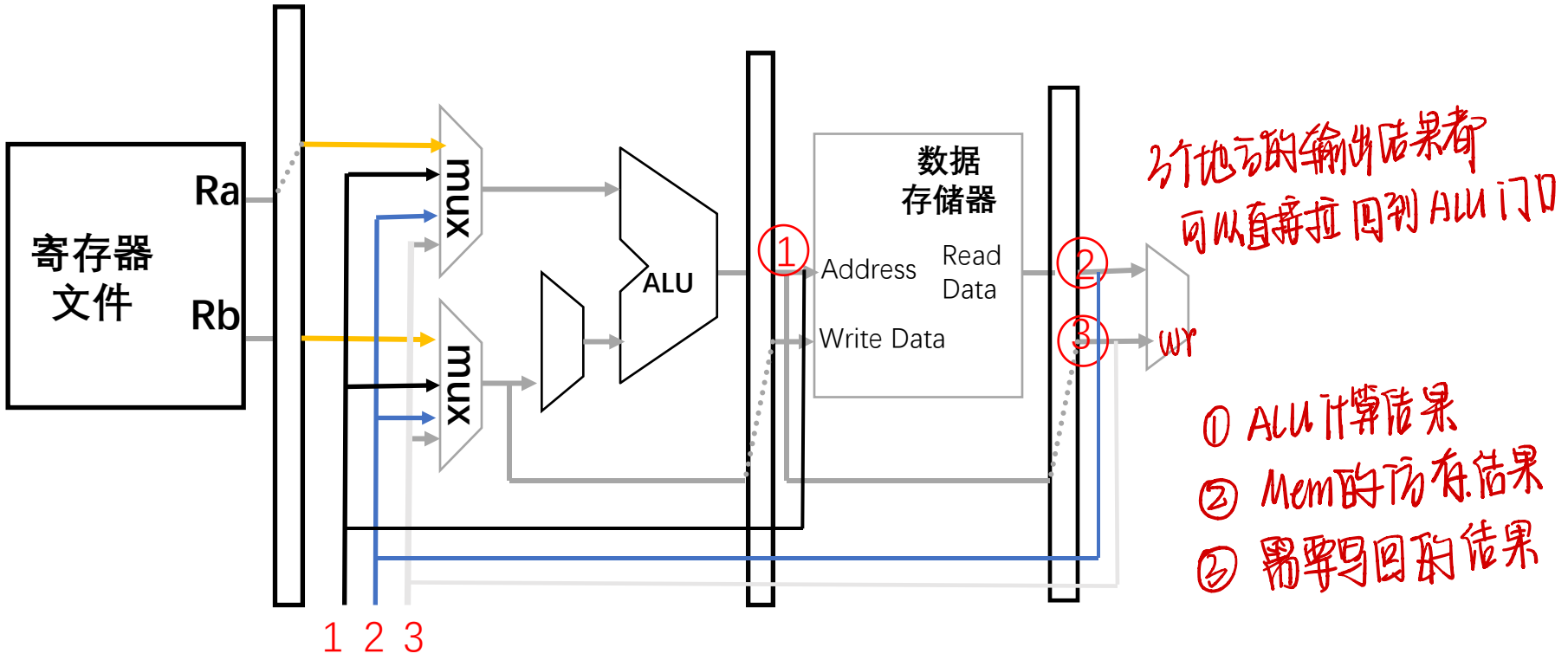
Compare the source registers of the instruction in the decode stage with the destination register of the uncommitted instructions.

另一种硬件解决方案： 前向传递（forwarding）



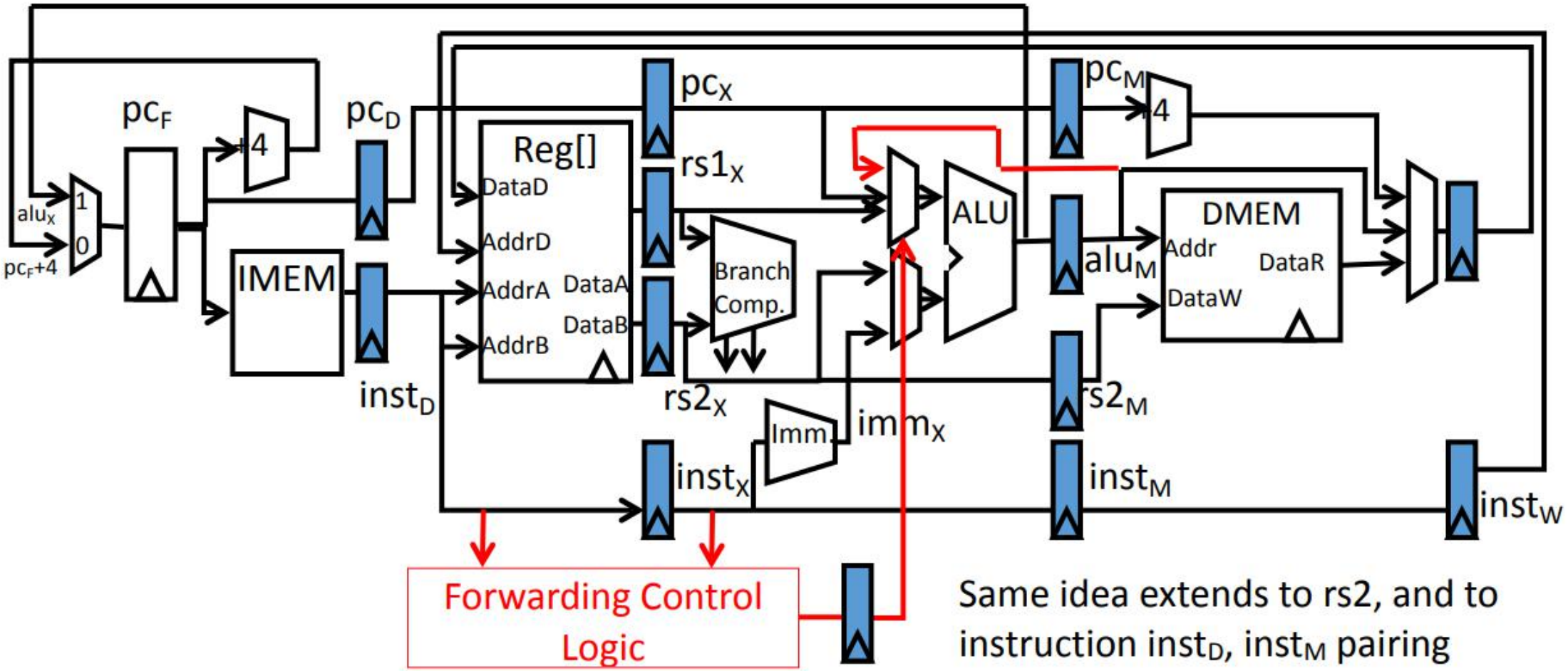
不等写回寄存器，就将产生的结果直接传送到当前周期需要结果的功能单元(例如： ALU)的输入端

前向传递（Forwarding）

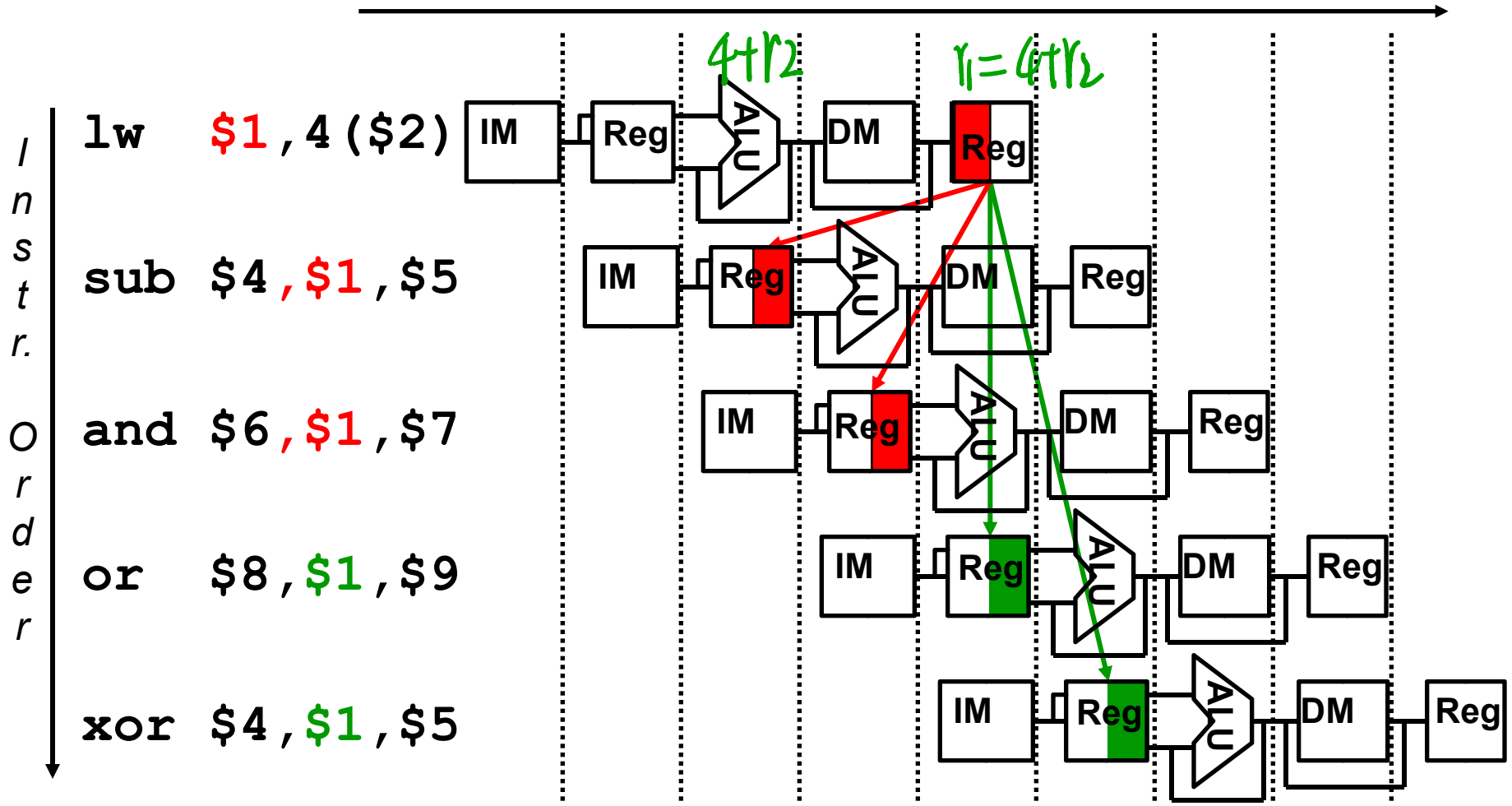


- 将前面指令的结果直接从其产生的地方定向到当前指令所需的位置
- 功能单元的输出不仅可以定向到其自身的输入，而且还可以定向到其它单元的输入

举例： Forwarding 的具体实现

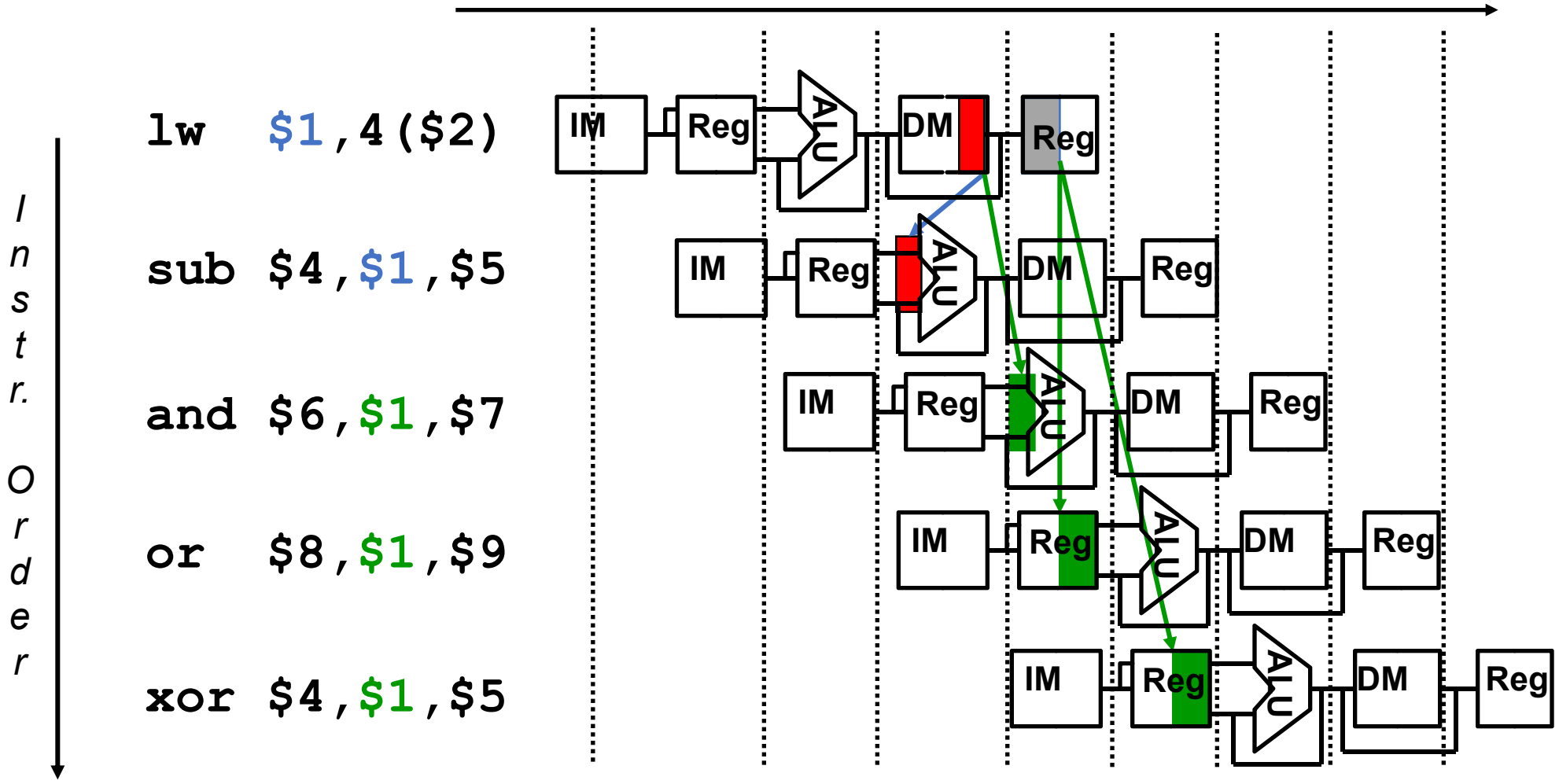


读存储器导致数据冒险

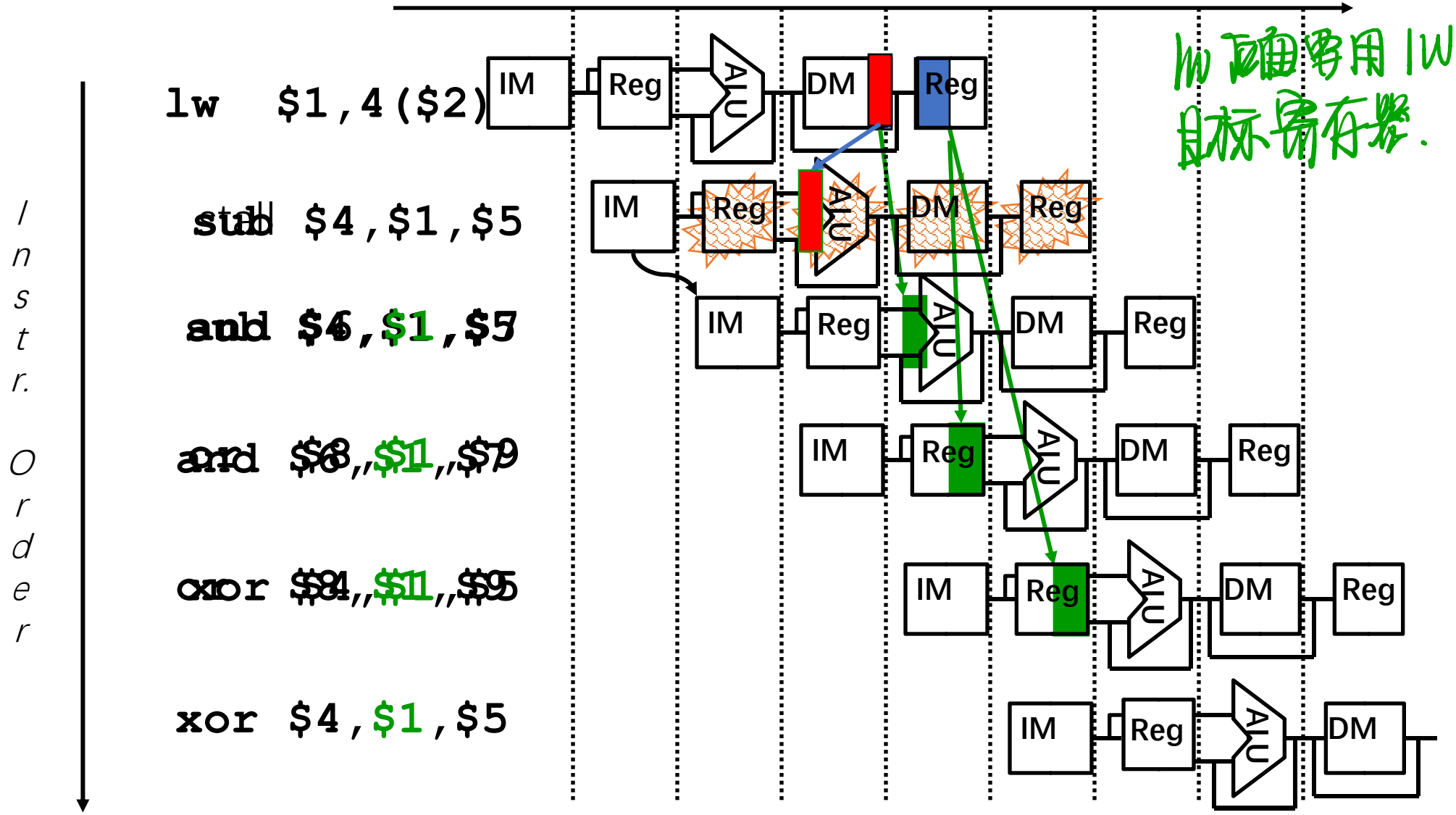


- “读存储器-使用” 冒险（Load-use data hazard）

用前向通路能解决“读存储器-使用”冒险吗？



前向通路不能解决“读存储器-使用”冒险

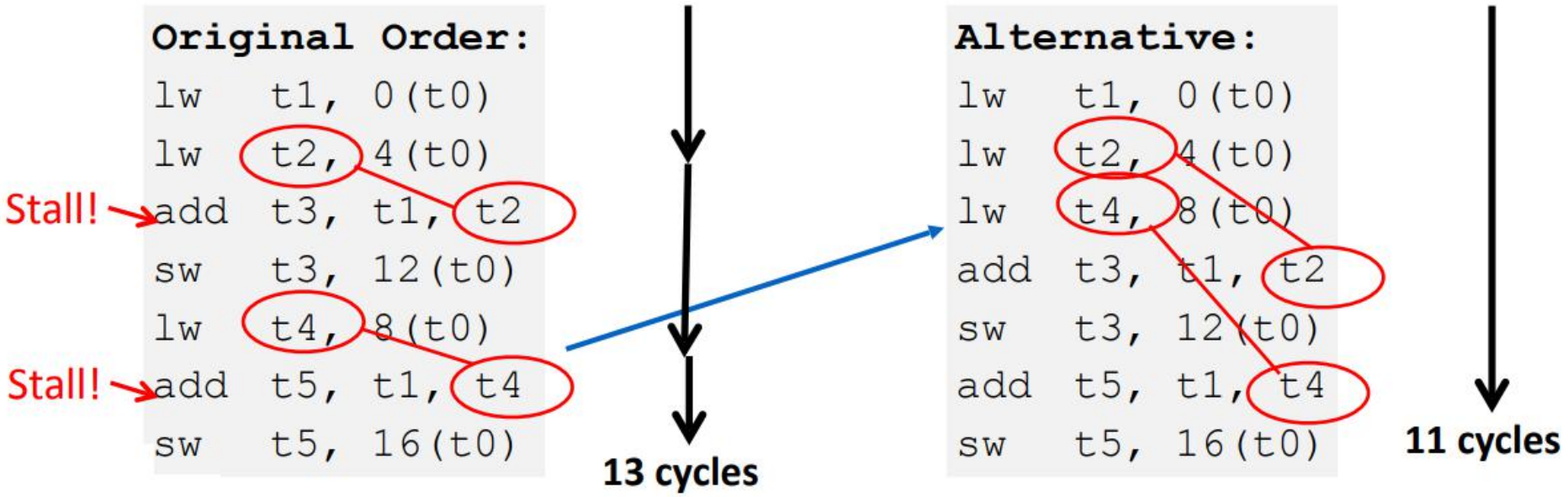


- 仍然需要一个周期的停顿

Code Scheduling to Avoid Stalls

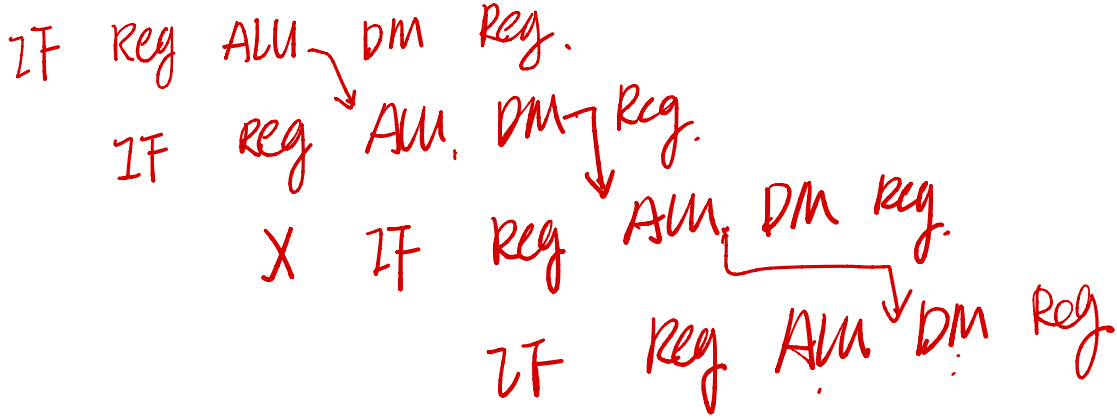
- Reorder code to avoid use of load result in the next instruction!
- RISC-V code for `D=A+B;  E=A+C;`

lw和add之间间隔1条指令



单项选择：以下一段RISC-V指令序列

```
add  x15, x12, x11      #R[x15] ⟵ R[x12]+R[x11]
ld   x13, 4(x15)        #R[x13] ⟵ M[R[x15]+4]
sub  x14, x15, x13      #R[x14] ⟵ R[x15]-R[x13]
sd   x14, 0(x15)        #R[x14] ⟶ M[R[x15]+0]
```



假定采用“取指、译码/取数、执行、访存、写回”这5段流水线方式，采用“转发（前向通路，forwarding, bypassing）”技术；时钟周期的长度足够让写寄存器发生在一个周期的上半段、读寄存器发生在一个周期的后半段；需要总共加入_____条空操作（nop）指令才能使这段程序不发生数据冒险。

A . 0　　B. 1　　C. 2　　D.3

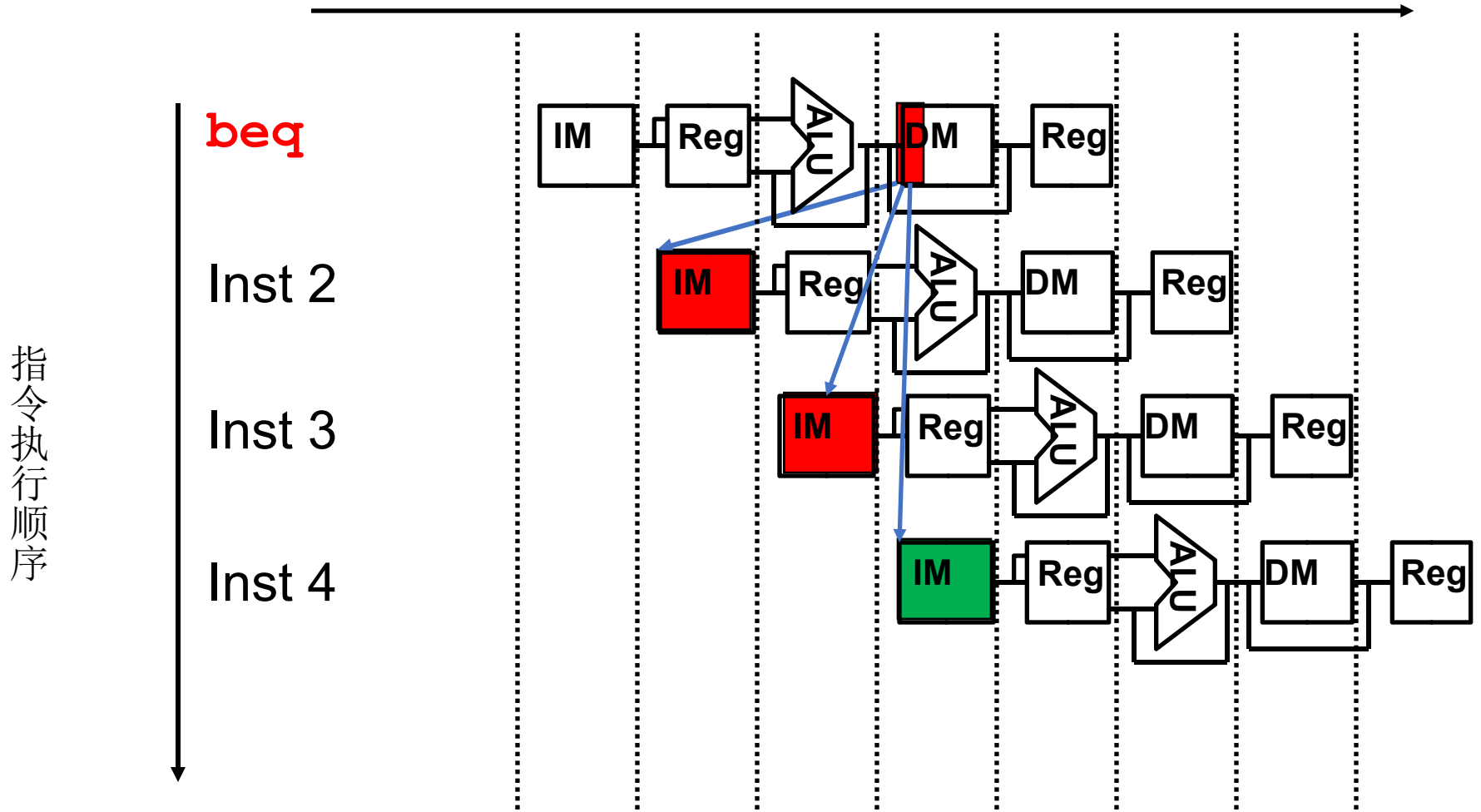
控制冒险

控制冒险

- 控制冒险（control hazards）
  - 由转移指令引起
    - 无条件转移 (j, jal, jalr)： 跳转到指定位置
    - 条件转移 (beq, bne,blt, bge)
      - 转移成功：将PC值改变为转移目标地址

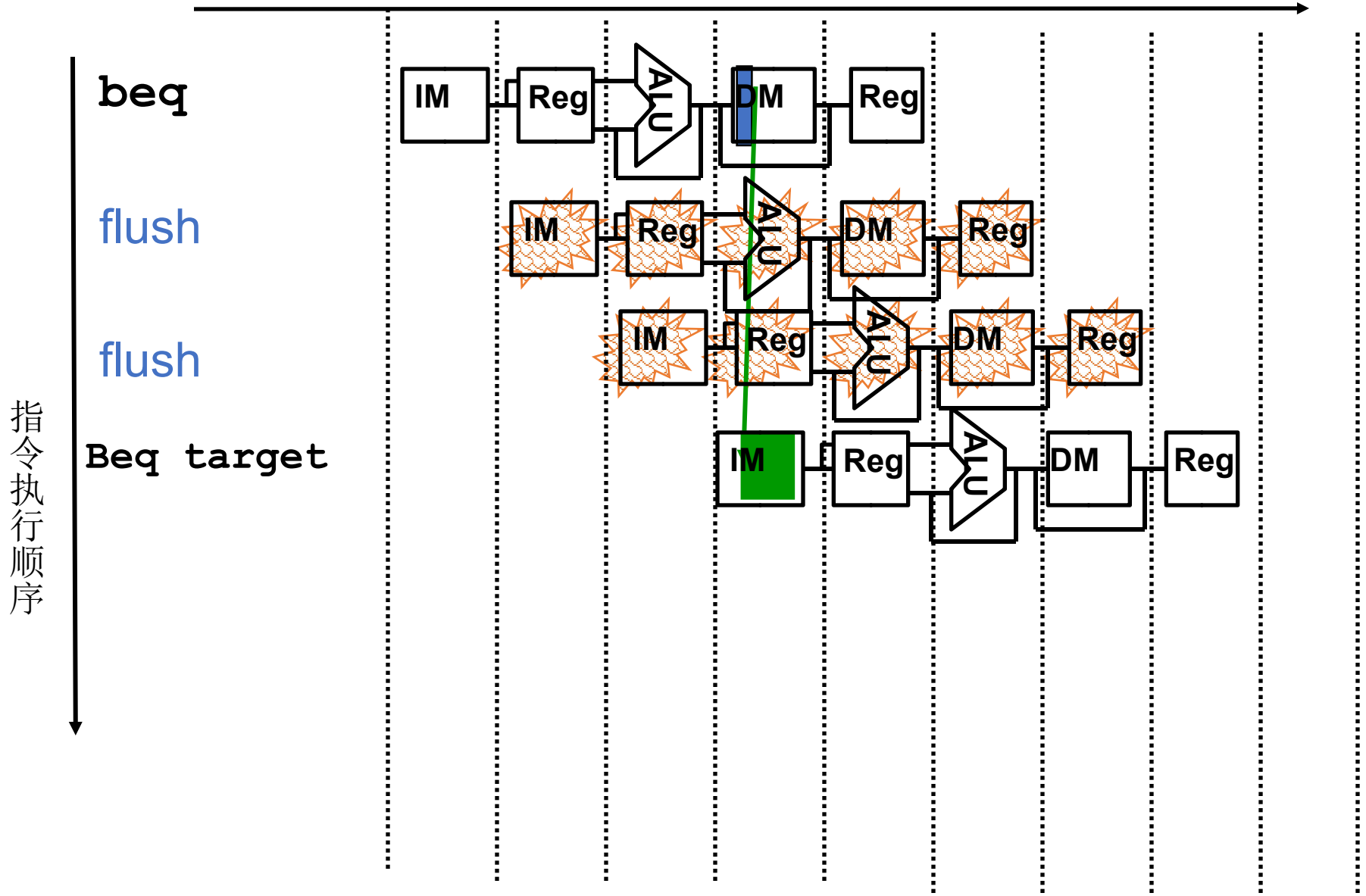
当转移指令的结果还未确定定时，无法决定下一步执行哪条指令。

转移指令（分支指令）引发控制冒险

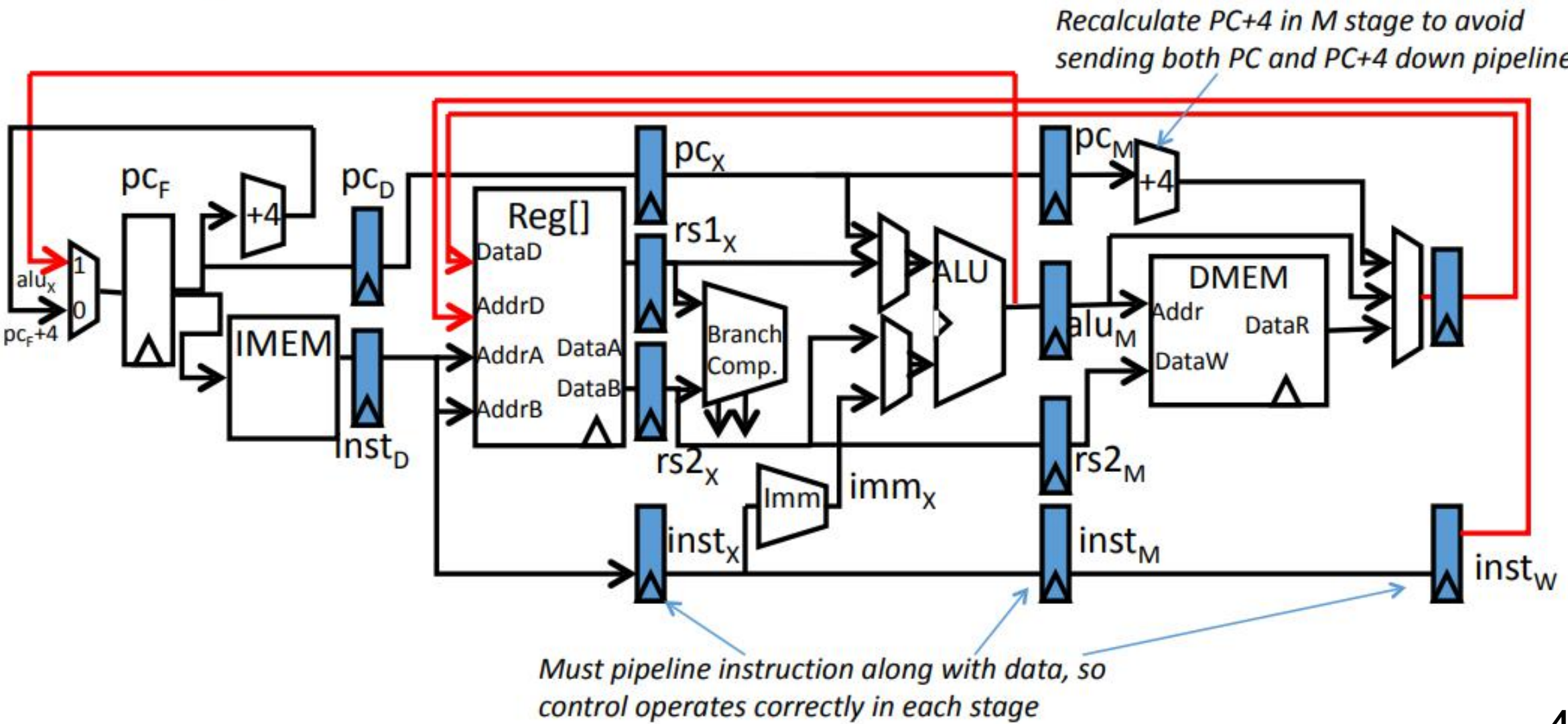


- 新 PC 的值要在MEM 段才写入， 引发了控制冒险

解决控制冒险最简单的方法：清空



Pipelined RISC-V RV32I Datapath



控制冒险对CPI的影响

- 指令流水线 :理想CPI＝1
- 若条件转移指令（分支指令）的频率为30%,
- 条件转移语句：如果转移会导致浪费2个时钟周期

  则：实际CPI（cycle per instr.）＝1＋30%×2≈ 1.6

- 控制冒险会严重影响CPI
  - 流水段越长，影响越大
  - Pentium 3：转移开销 10周期
  - Pentium 4： 转移开销 20周期

谢谢！