

第3章 流水线模型机

- ❖ C. 1 流水线的基本概念
- ❖ C. 2 流水线的主要障碍——流水线冒险
- ❖ C. 3 流水线处理机设计
- ❖ C. 4 异常事件处理
- ❖ C. 5 扩展流水线到多周期操作

C.2 流水线的主要障碍—流水线冒险

❖一、冒险分类与有停顿流水线性能

❖二、结构冒险

❖三、数据冒险

❖四、控制冒险

一、冒险分类与有停顿流水线性能

❖回顾：

- 流水线

- 以**重叠方式**执行指令，制造**快速CPU**的实现技术(减少 CPUtime, 改进吞吐量)

- 流水线理想加速比：**流水线的段数**

- 从执行每条指令执行用多周期的机器角度：
 - 流水线减少了**CPI**.

冒险分类

❖ 冒险分类

■ 结构冒险

- 指令重叠执行时，发生硬件资源冲突

■ 数据冒险

- 几条指令重叠执行时，后面指令依赖前面指令的结果却没有准备好（还没有计算或存储）

■ 控制冒险：流水线执行转移指令时

- 在进入ID段时，转移条件和转移目标地址不能按时提供给IF段取下一条指令。

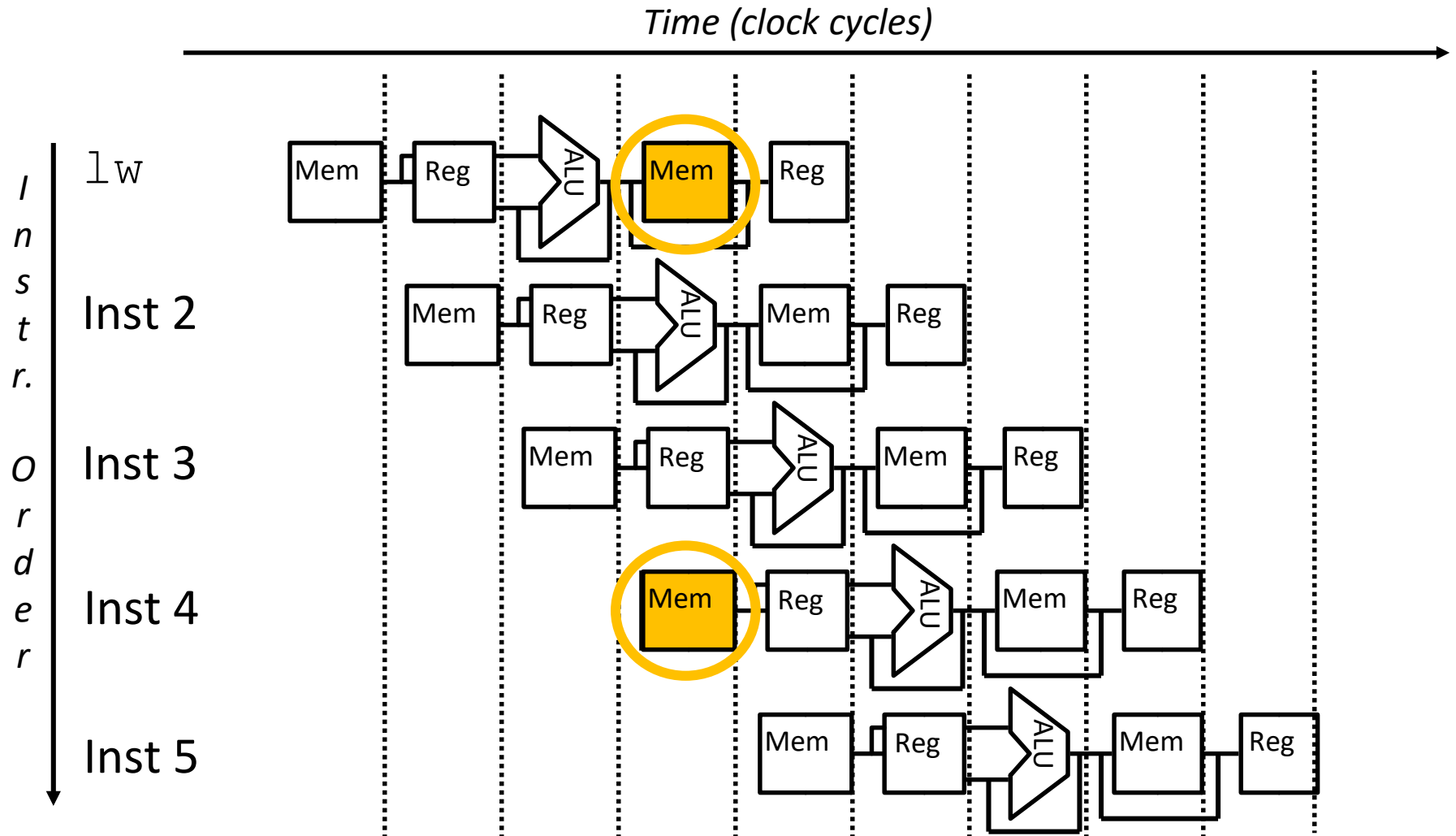
❖ 冒险出现时：避免流水线上有冒险的指令执行下一个流水段

二、结构冒险：流水段竞争

❖ 结构冒险

- 发生在同一个时钟周期，2条或多条指令想要使用同一个硬件资源
- 引起流水线机器停顿
- 常见引起结构冒险的情况：
 - 多重访问存储器
 - 多重访问寄存器堆
 - 没有或没有充分流水功能部件

结构冒险



冒险总是可以用停顿解决

- ❖ 解决冒险最简单的方式就是**停顿流水线**
- ❖ 停顿意味着为某些指令暂停流水线一个或多个时钟周期。
- ❖ 一条指令被**停顿**后，它和其后的所有指令被**停顿**；该指令之前的指令继续执行。
- ❖ 一个流水线**停顿**也称为**流水线气泡或气泡**。
- ❖ **停顿**时，没有新的指令被取到流水线。

插入停顿 Stall

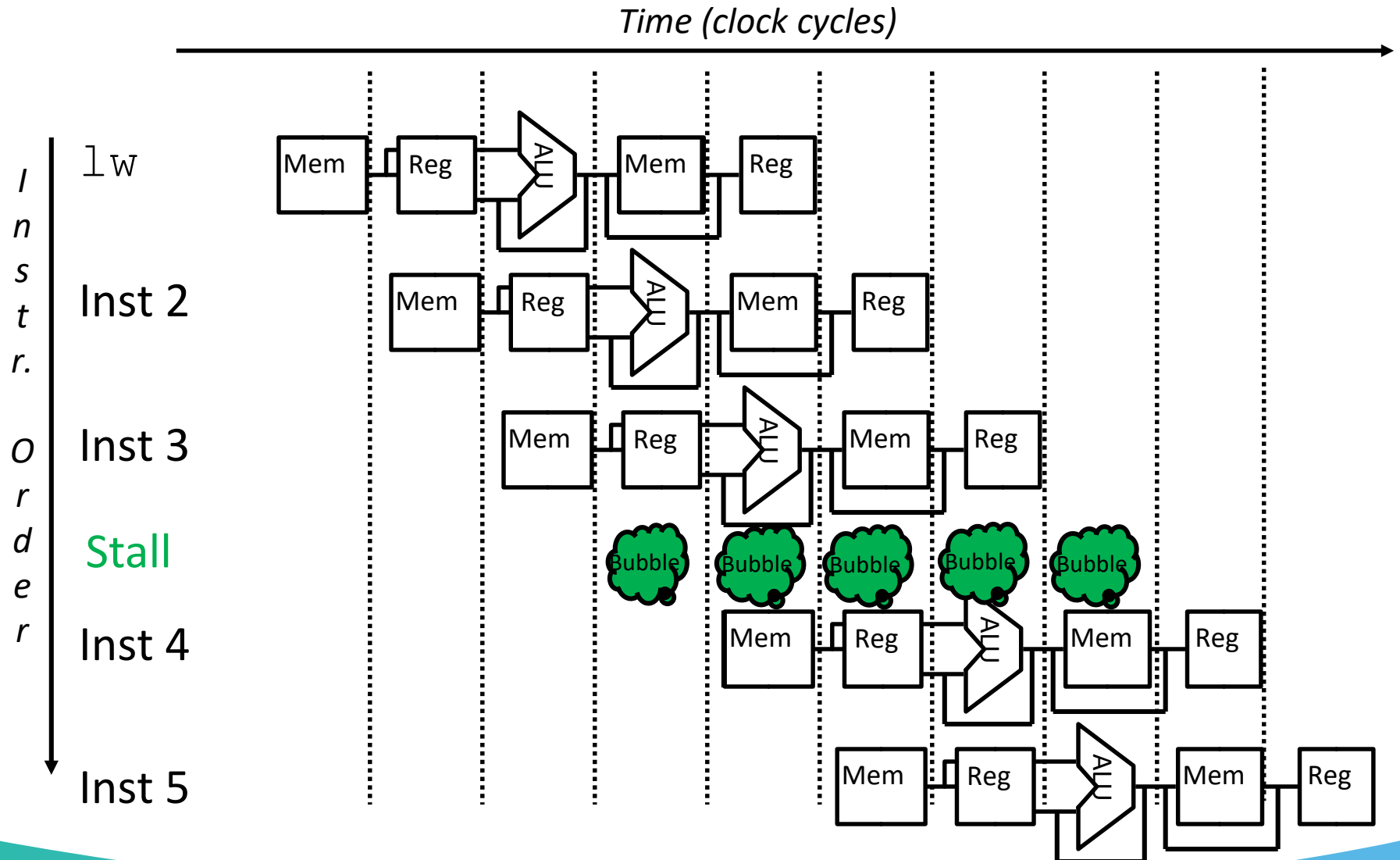


Figure C.5

	Clock cycle number									
Instruction	1	2	3	4	5	6	7	8	9	10
Load instruction	IF	ID	EX	MEM	WB					
Instruction $i + 1$		IF	ID	EX	MEM	WB				
Instruction $i + 2$			IF	ID	EX	MEM	WB			
Instruction $i + 3$				Stall	IF	ID	EX	MEM	WB	
Instruction $i + 4$						IF	ID	EX	MEM	WB
Instruction $i + 5$							IF	ID	EX	MEM
Instruction $i + 6$								IF	ID	EX

Figure C.5 A pipeline stalled for a structural hazard—a load with one memory port. As shown here, the load instruction effectively steals an instruction-fetch cycle, causing the pipeline to stall—no instruction is initiated on clock cycle 4 (which normally would initiate instruction $i + 3$). Because the instruction being fetched is stalled, all other instructions in the pipeline before the stalled instruction can proceed normally. The stall cycle will continue to pass through the pipeline, so that no instruction completes on clock cycle 8. Sometimes these pipeline diagrams are drawn with the stall occupying an entire horizontal row and instruction 3 being moved to the next row; in either case, the effect is the same, since instruction $i + 3$ does not begin execution until cycle 5. We use the form above, since it takes less space in the figure. **Note that this figure assumes that instructions $i + 1$ and $i + 2$ are not memory references.**

Figure C.5

	Clock cycle number									
Instruction	1	2	3	4	5	6	7	8	9	10
Load instruction	IF	ID	EX	MEM	WB					
Instruction $i + 1$		IF	ID	EX	MEM	WB				
Instruction $i + 2$			IF	ID	EX	MEM	WB			
Instruction $i + 3$				Stall	IF	ID	EX	MEM	WB	
Instruction $i + 4$						IF	ID	EX	MEM	WB
Instruction $i + 5$							IF	ID	EX	MEM
Instruction $i + 6$								IF	ID	EX

Figure C.5 A pipeline stalled for a structural hazard—a load with one memory port. As shown here, the load instruction effectively steals an instruction-fetch cycle, causing the pipeline to stall—no instruction is initiated on clock cycle 4 (which normally would initiate instruction $i + 3$). Because the instruction being fetched is stalled, all other instructions in the pipeline before the stalled instruction can proceed normally. The stall cycle will continue to pass through the pipeline, so that no instruction completes on clock cycle 8. Sometimes these pipeline diagrams are drawn with the stall occupying an entire horizontal row and instruction 3 being moved to the next row; in either case, the effect is the same, since instruction $i + 3$ does not begin execution until cycle 5. We use the form above, since it takes less space in the figure. **Note that this figure assumes that instructions $i + 1$ and $i + 2$ are not memory references.**

由于访问内存使用同一个端口，资源冲突导致结构险象，使得流水线停顿。

如表中所示，由于第4号周期无法同时进行数据的读写和指令的读取，在4号时钟周期没有启动指令（它通常应该启动指令 $i+3$ ）。因为读指令停顿，所以流水线中位于停顿指令之前的所有其他指令都可以正常进行。由停顿周期将继续通过流水线，所以在8号时钟周期中没有完成指令 $i+3$ 。有时在绘制这些流水线表时，让流水线占据整个水平行，指令 $i+3$ 被移到下一行；无论采用哪种画法，效果都是一样的，因为指令 $i+3$ 直到5号周期才开始执行。因为以上形式占据的空间较少，所以我们采用了这一形式。 **注意，本图假定指令 $i+1$ 和 $i+2$ 不读写数据存储器。**

补充：本图也需要假定指令 $i+3$ 不读写数据存储器。

有停顿的流水线性能

- ❖ 停顿会降低流水线的性能，使其性能比理想的差
- ❖ 回忆加速比公式：

$$\begin{aligned}\text{流水线加速比} &= \frac{\text{非流水线指令平均执行时间}}{\text{流水线指令平均执行时间}} \\ &= \frac{\text{非流水线CPI} \times \text{非流水线时钟周期}}{\text{流水线CPI} \times \text{流水线时钟周期}} \\ &= \frac{\text{非流水线CPI}}{\text{流水线CPI}} \times \frac{\text{非流水线时钟周期}}{\text{流水线时钟周期}}\end{aligned}$$

流水线实现的情况

- ❖ 假设流水线处理器的理想CPI 是1。

$$\begin{aligned}\text{流水线CPI} &= \text{理想CPI} + \text{平均每条指令的停顿周期数} \\ &= 1 + \text{平均每条指令的停顿周期数}\end{aligned}$$

- ❖ 忽略流水线时钟周期的额外开销。
- ❖ 假设流水段是理想平衡的。

$$\text{流水线加速比} = \frac{\text{流水线级数}}{1 + \text{平均每条指令的停顿周期数}}$$

Example

Suppose that data references constitute 40% of the mix, and that the ideal CPI of the pipelined processor, ignoring the structural hazard, is 1. Assume that the processor with the structural hazard has a clock rate that is 1.05 times higher than the clock rate of the processor without the hazard. Disregarding any other performance losses, is the pipeline with or without the structural hazard faster, and by how much?

假定数据读写占总体的40%，流水化处理器的理想CPI为1（忽略结构冒险）。假定与没有冒险的处理器相比，有结构冒险处理器的时钟频率为其1.05倍。不考虑所有其他性能损失，有结构冒险和无结构冒险相比，哪种流水线更快？快多少？

Example

Suppose that data references constitute 40% of the mix, and that the ideal CPI of the pipelined processor, ignoring the structural hazard, is 1. Assume that the processor with the structural hazard has a clock rate that is 1.05 times higher than the clock rate of the processor without the hazard. Disregarding any other performance losses, is the pipeline with or without the structural hazard faster, and by how much?

Example

IC: Instruction Count

Suppose that data references constitute **40%** of the mix, and that the ideal CPI of the pipelined processor, ignoring the structural hazard, is 1. Assume that the processor with the structural hazard has a clock rate that is **1.05** times higher than the clock rate of the processor without the hazard. Disregarding any other performance losses, is the pipeline with or without the structural hazard faster, and by how much?

Answer There are several ways we could solve this problem. Perhaps the simplest is to compute the average instruction time on the two processors:

$$\text{Average instruction time} = \text{IC} \times \text{CPI} \times \text{Clock cycle time}$$

Since it has no stalls, the average instruction time for the ideal processor is simply the $\text{IC} \times \text{Clock cycle time}_{\text{ideal}}$. The average instruction time for the processor with the structural hazard is

$$\begin{aligned} \text{Average instruction time} &= \text{IC} \times \text{CPI} \times \text{Clock cycle time} \\ &= \text{IC} \times (1 + 0.4 \times 1) \times \frac{\text{Clock cycle time}_{\text{ideal}}}{1.05} \\ &= \text{IC} \times 1.33 \times \text{Clock cycle time}_{\text{ideal}} \end{aligned}$$

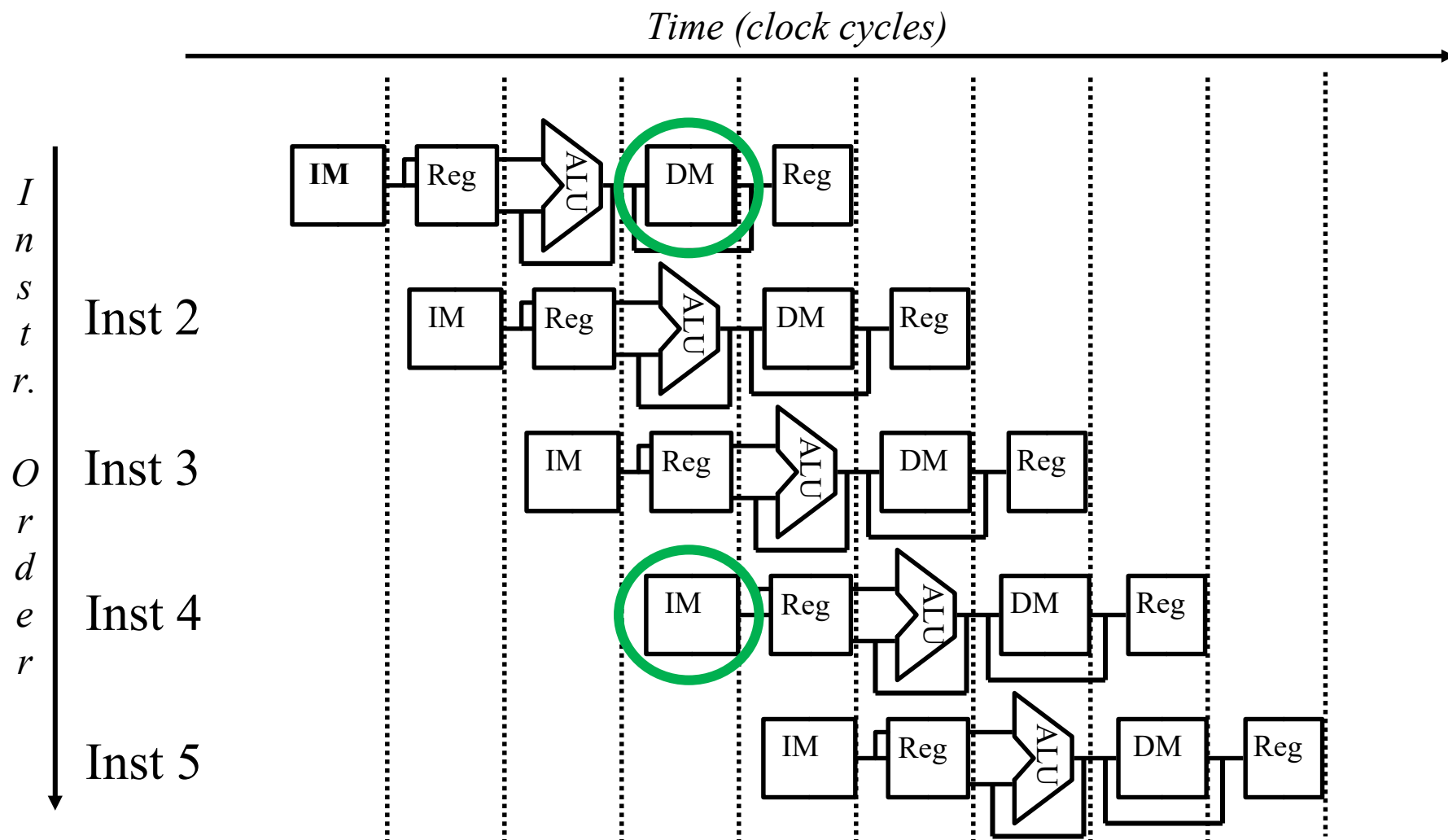
Clearly, the processor without the structural hazard is faster; we can use the ratio of the average instruction times to conclude that the processor without the hazard is 1.33 times faster.

显然，没有结构冒险的处理器更快一些；根据平均指令时间的比值，我们可以得出结论，无冒险处理器的速度快1.33倍。

结构冒险的消除

- ❖ 为了避免因为存储器引起的出现这种结构冒险，可以为指令提供独立的存储器访问，既可以将缓存分为独立的指令缓存和数据缓存，也可以使用一组缓冲区来保存指令，这种缓冲区通常称为指令缓冲区。

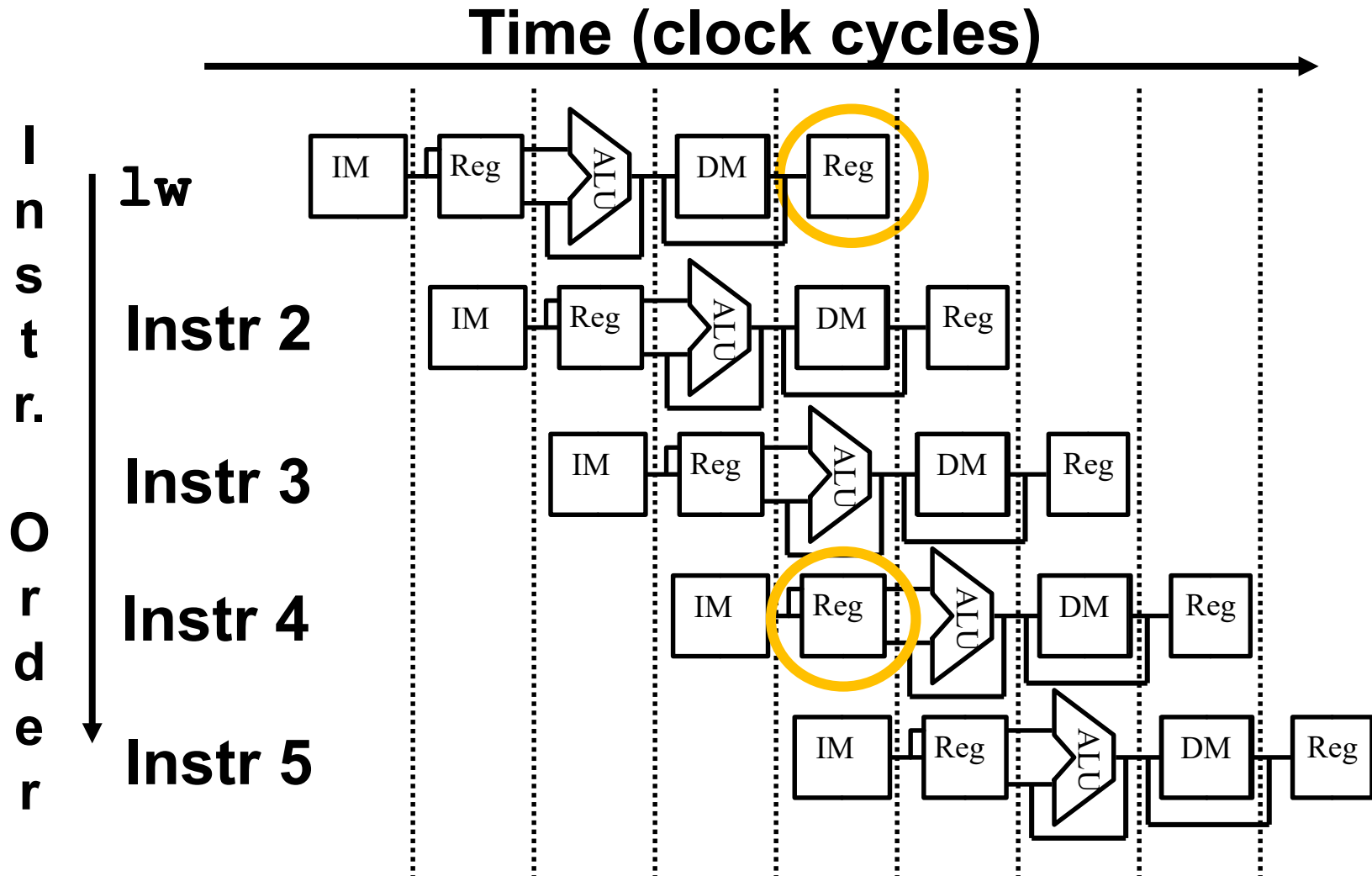
分开的指令和数据存储器



- 分开的指令和数据存储器/ 多个存储器端口/ 指令缓冲器 :

取指令和存取数据使用不同的硬件资源

Structural Hazard #2: Registers (1/2)



Can we read and write to registers simultaneously?

Structural Hazard #2: Registers (2/2)

❖ Solution has been used:

1) RegFile access is *VERY* fast: takes less than half the time of ALU stage

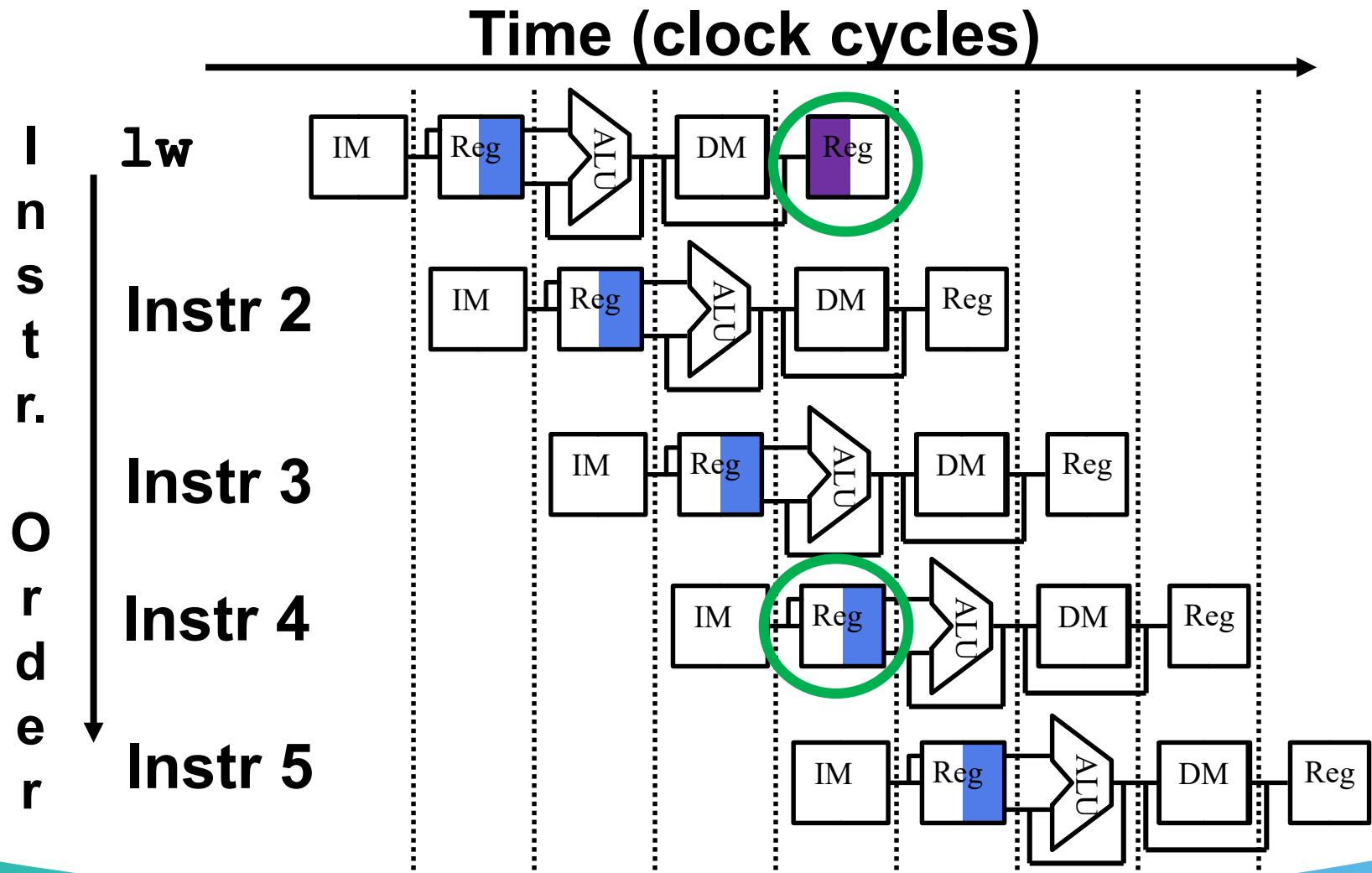
- **Write** to Registers during **first half** of each clock cycle
- **Read** from Registers during **second half** of each clock cycle

2) Build RegFile with **independent** read and write ports

❖ **Result: can perform Read and Write during same clock cycle**

Write /Read Registers in single cycle

(In Reg, right half highlight read, left half write)



没有完全流水化的功能部件： 可能引起结构冒险

Unpipelined Float Adder

ADDD	IF	ID	ADDD						WB	
ADDD		IF	ID	stall	stall	stall	stall	stall	ADDD	

Not fully pipelined Float Adder

ADDD	IF	ID	A1		A2		A3		WB	
ADDD		IF	ID	stall	A1		A2		A3	

Fully pipelined Float Adder

ADDD	IF	ID	A1	A2	A3	A4	A5	A6	WB	
ADDD		IF	ID	A1	A2	A3	A4	A5	A6	WB

为什么允许结构冒险？

❖ 通常应消除结构冒险，但因下列的原因可以允许存在结构冒险

❖ 减少成本

- 如，增加分开的caches需要两倍的存储器带宽
- 完全流水浮点部件需要很多逻辑门
- 如果结构冒险不经常发生，且消除冒险的成本很高

❖ 减少部件延迟

- 制造流水的功能部件增加延迟
(流水线附加开销 -> 流水线寄存器)
- 非流水线部件的每个操作需要更少的时钟

结构冒险总结

❖ 冒险分类

■ 结构冒险

- 硬件资源冲突
- 解决：可以增加硬件资源；
或者功能部件完全流水；
否则，只有停顿流水线

■ 数据冒险

- 几条指令重叠执行时，一条指令依赖前面指令的结果却没有准备好（还没有计算或存储）

■ 控制冒险

- 在进入ID段时，转移条件和转移目标地址，不能按时提供给IF段取指令

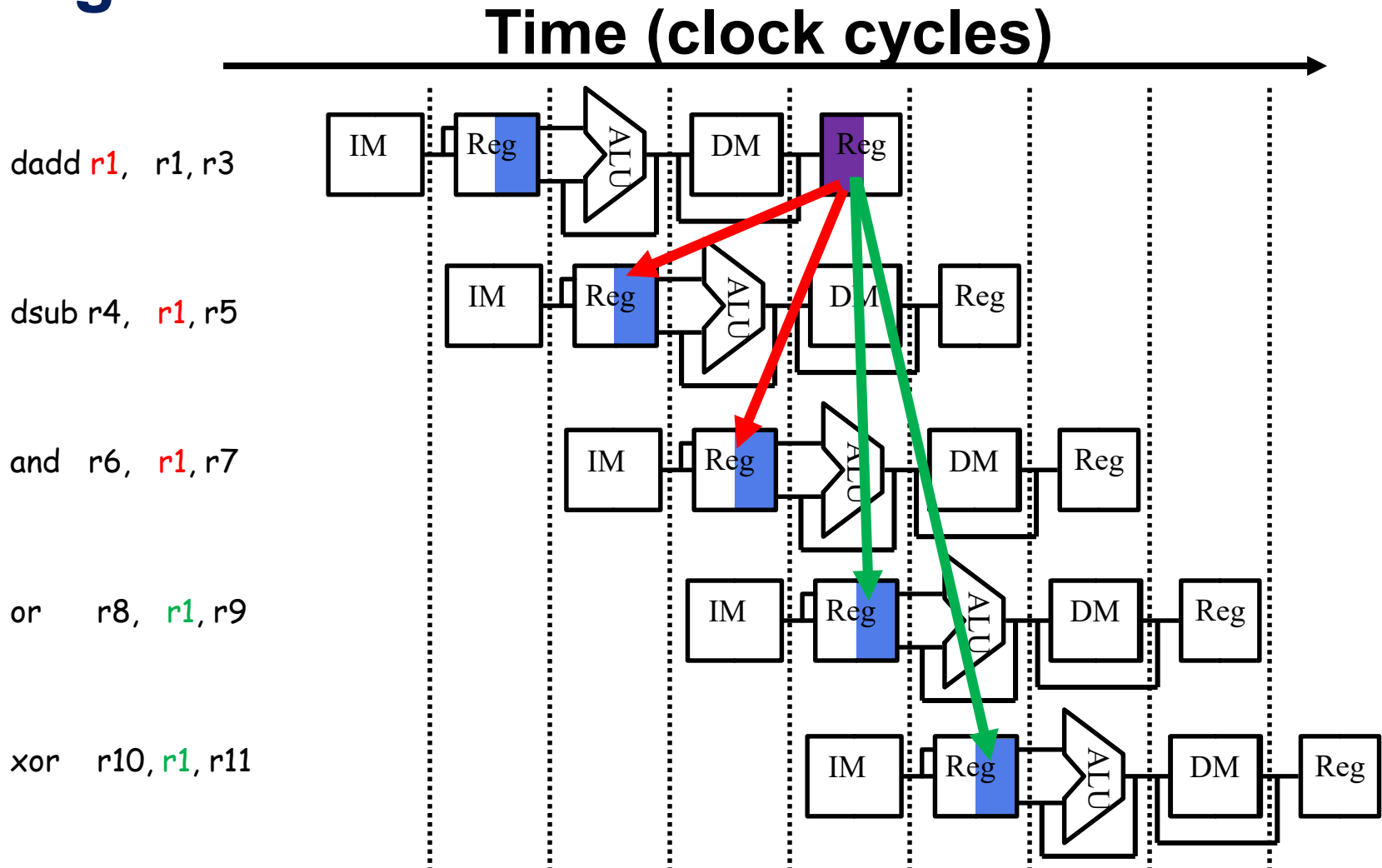
三、数据冒险

- ❖ 数据冒险：由于流水线上指令重叠执行，使得*后面依赖前面指令的结果，得不到前面指令的结果。*
- ❖ 数据相关：考虑两条指令i和j，i在j的前面，如果下述条件之一成立，则称指令j与指令i数据相关：
 - ❖ (1) 指令j使用指令i产生的结果；
 - ❖ (2) 指令j与指令k数据相关，而指令k又与指令i数据相关。
- ❖ 例子：

```
dadd r1, r1, r3
dsub r4, r1, r5
and  r6, r1, r7
or   r8, r1, r9
xor  r10, r1, r11
```


数据冒险

Figure C.6



数据冒险

❖ 基本结构

- 一条指令想要使用还没有“完成”的数据值
- “完成”表示“数据值已经被计算了”和“值已经被放好，正常情况可以在流水线硬件中找到它”

❖ 原因

- 习惯上总是假定采用纯串行指令执行模型
- 对于 $k \geq 1$ ，指令N在指令N+1之前完成
- 在相邻指令之间有依赖存在

❖ 结论

- 数据冒险---指令想要的数据值还没有完成，或者值没有在正确的地方

提出的解决方法

❖ 提出的解决方法

- 如果数据不存在就不要让指令在数据冒险时重叠执行
- 如果数据已经存在就用它。
- 如果后面的指令没有依赖，就提前执行。

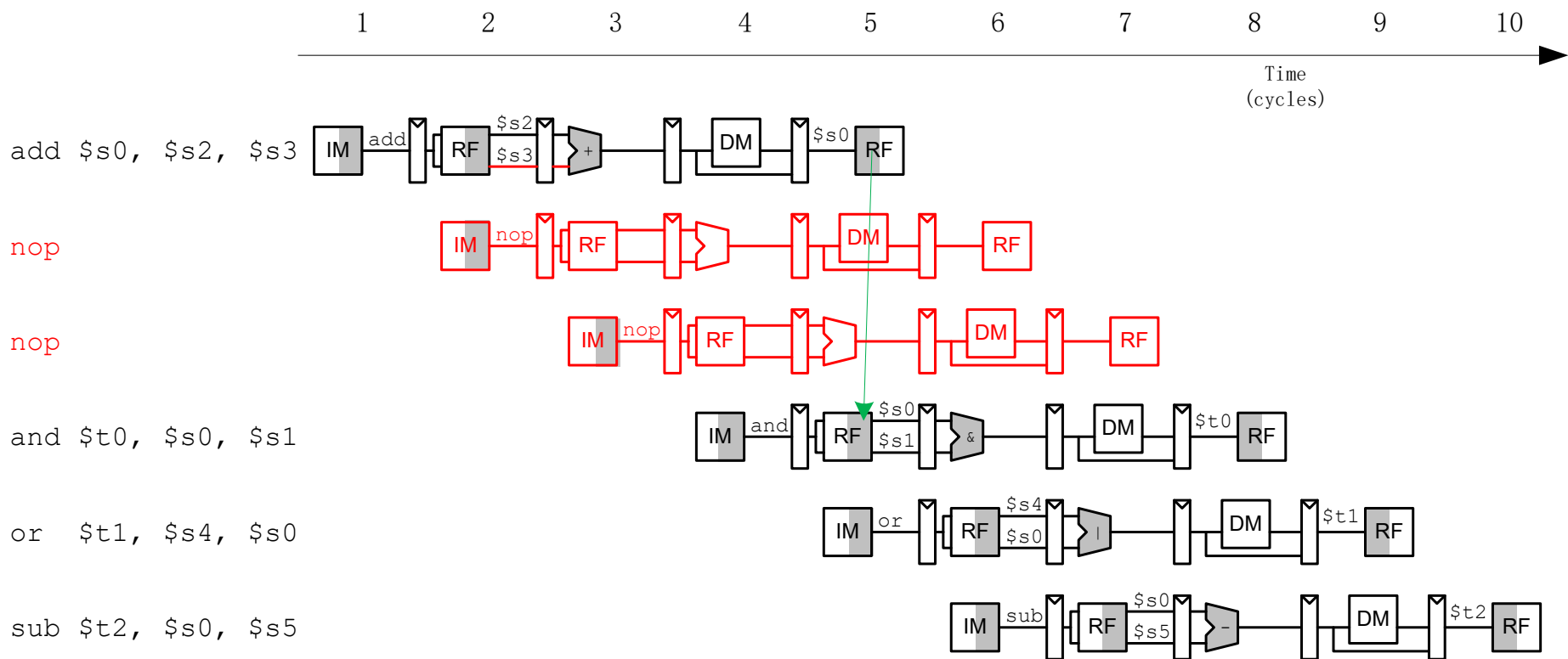
❖ 实施技术

- 如果数据不存在，就让软件插入nop指令，或由硬件让流水线停顿，直到冒险消除
- 如果数据已经存在就用它：超前查看

怎样停顿？

❖ 由编译器插入 **nop** 指令（软件）

Insert enough nops for result to be ready



怎样停顿？

❖ 增加硬件互锁（Interlock）！

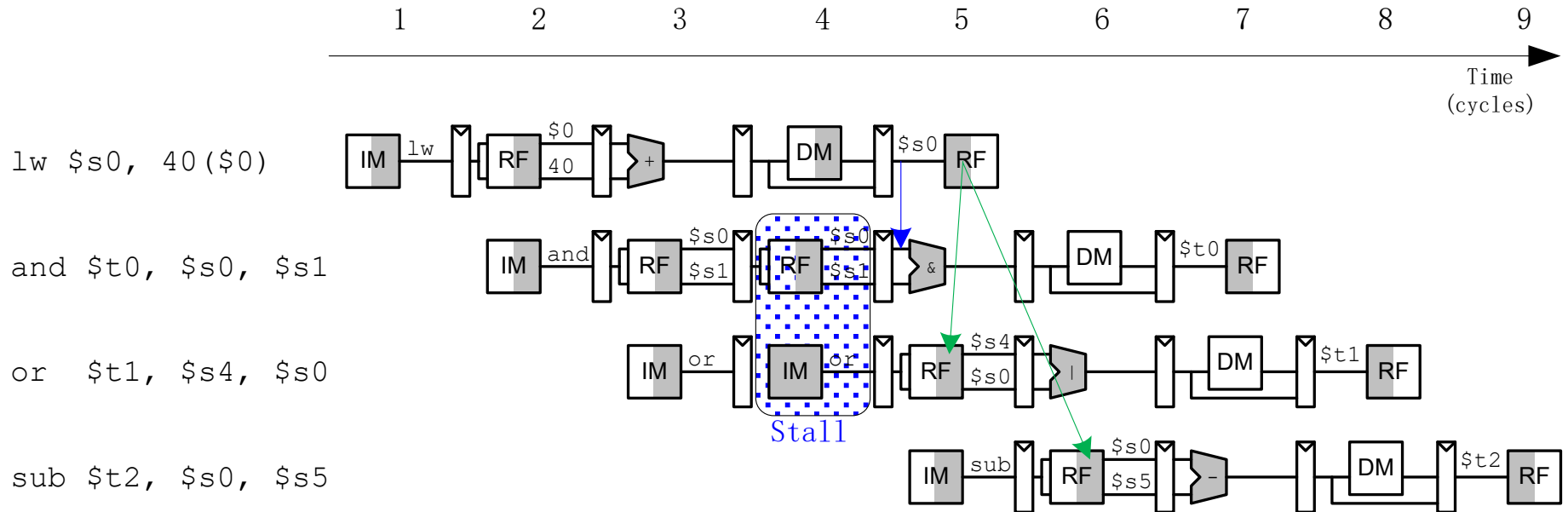
❖ 增加额外硬件检测需要停顿的情况

- 观察指令字段位（如，寄存器字段）
- 在流水线特定段检测是否有“先写后读”冲突

❖ 增加额外硬件放“气泡”（暂停）到流水线

- 实际操作：让有冒险的指令通过流水线，但是禁止允许任何结果写入机器状态的位
- 因此，有冒险的指令部分阶段当成nop指令

互锁： 插入 stalls（硬件）



Empty slots in the
pipe called bubbles;
means no real
instruction work
getting saved here

Forwarding

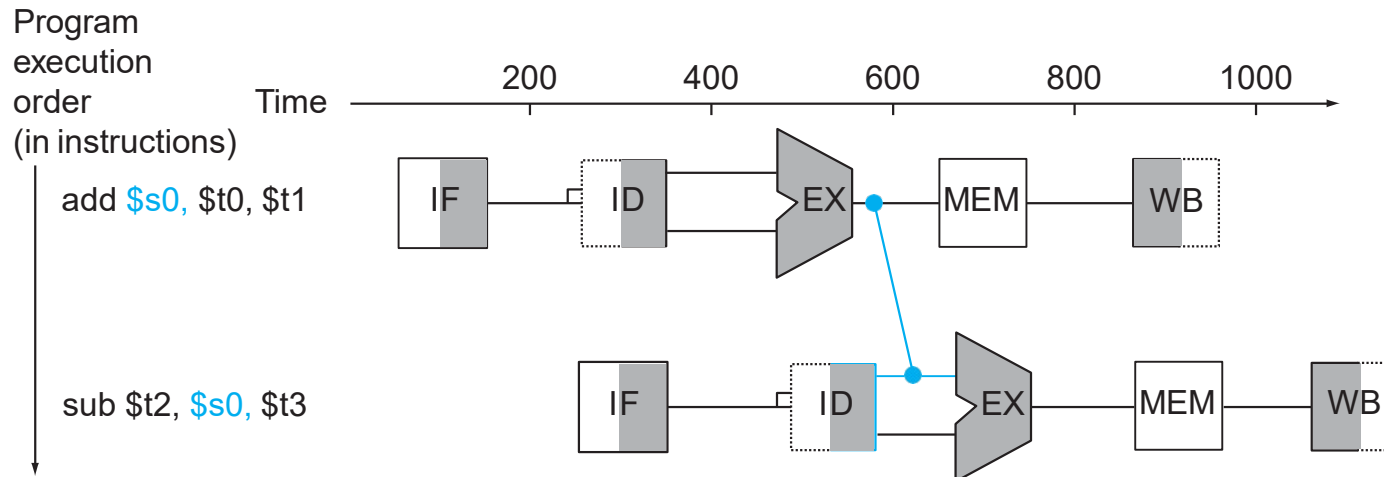
❖ 前推、直通、转发、超前查看

❖ 同义词: forwarding (bypass, short-circuiting)

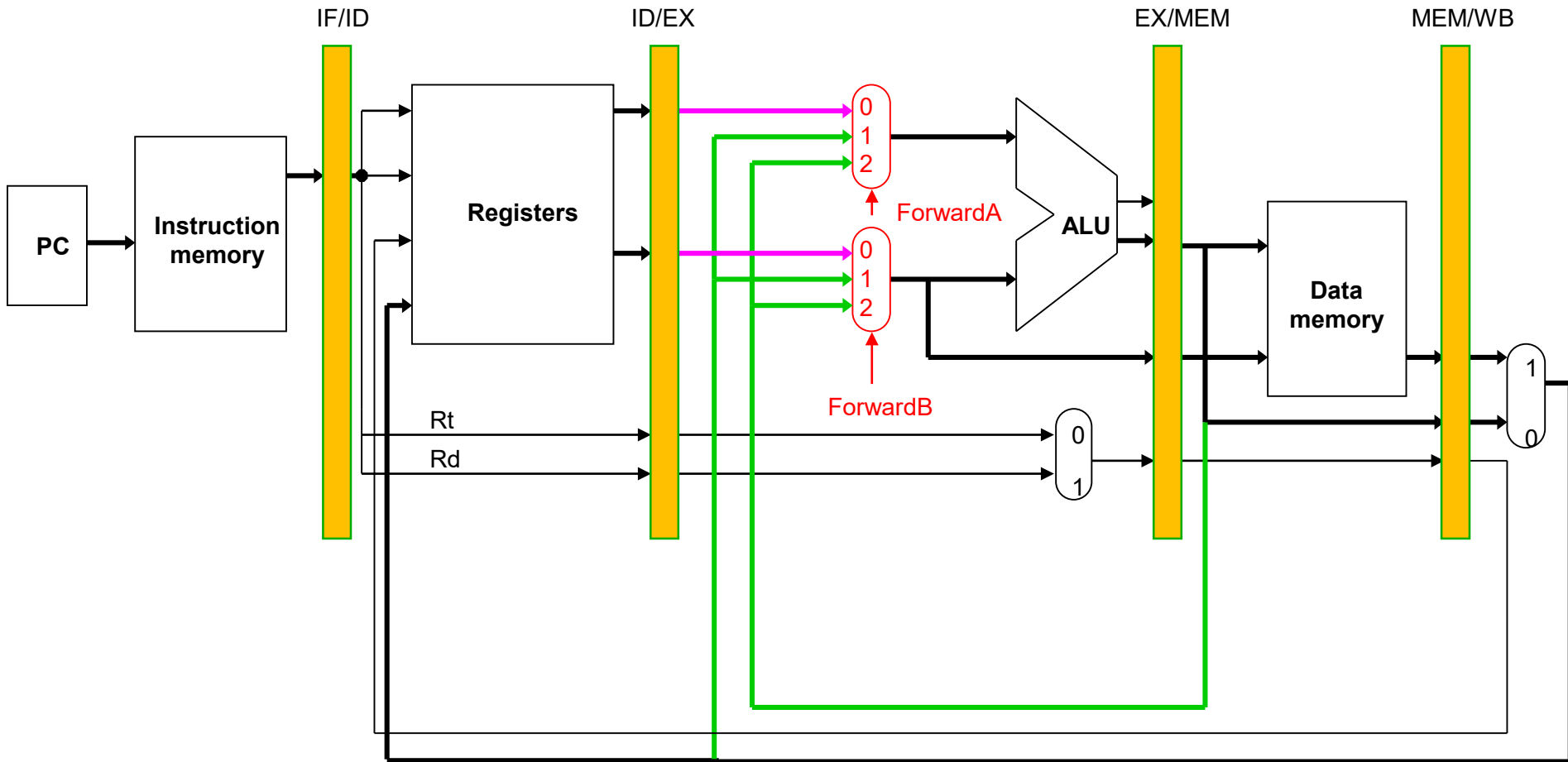
- 仔细观察，发现有冒险指令需要的结果可能已经计算出来，存放在流水线寄存器中
- 可以在数据通路上增加数据线（ buses ）传送这些结果
- 这些 buses 在数据通路中总是从后面的流水段连接到前面的流水段

直通技术：减少数据冒险停顿

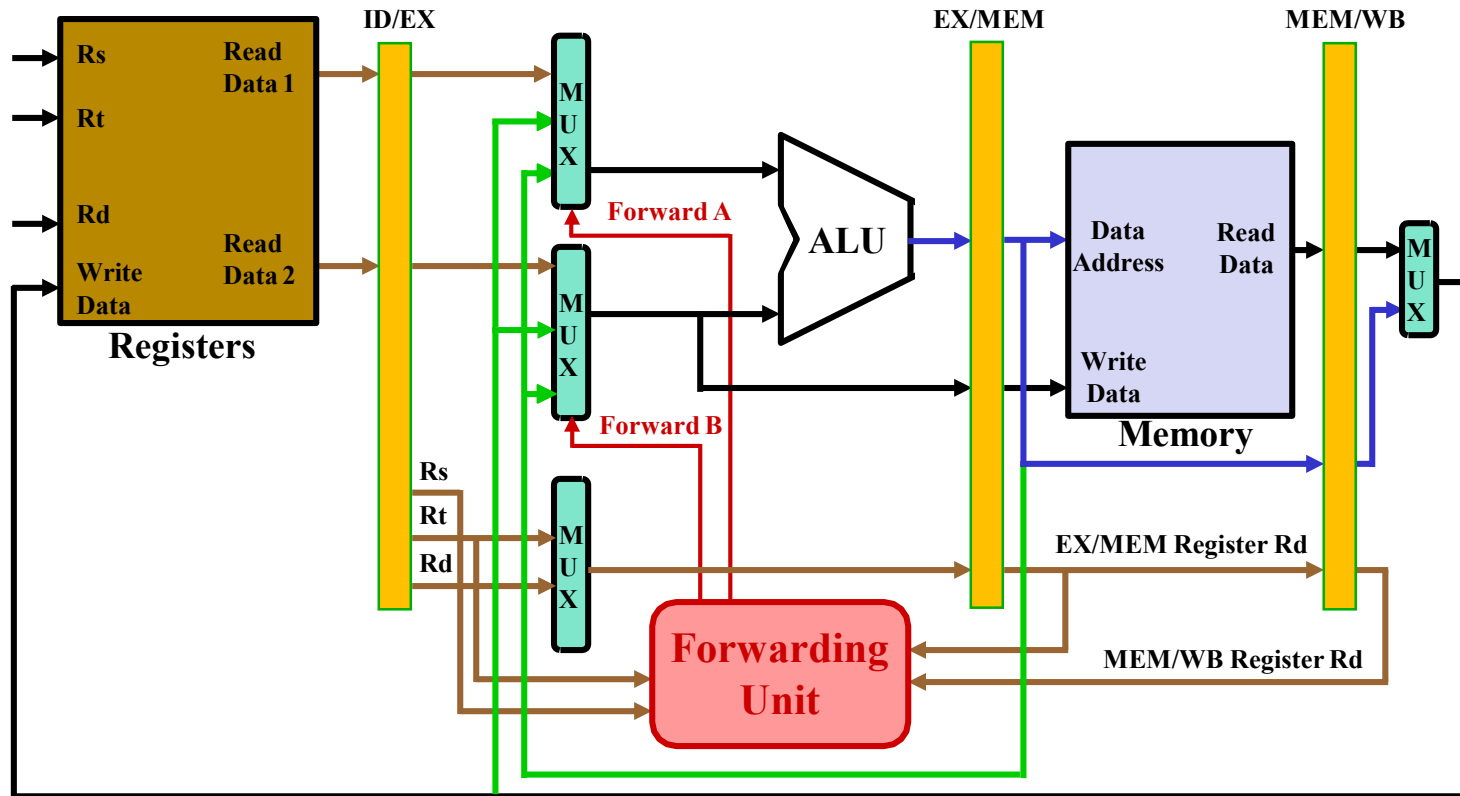
❖ 数据可能已经计算好—只是不在寄存器堆中



Forwarding的硬件变化



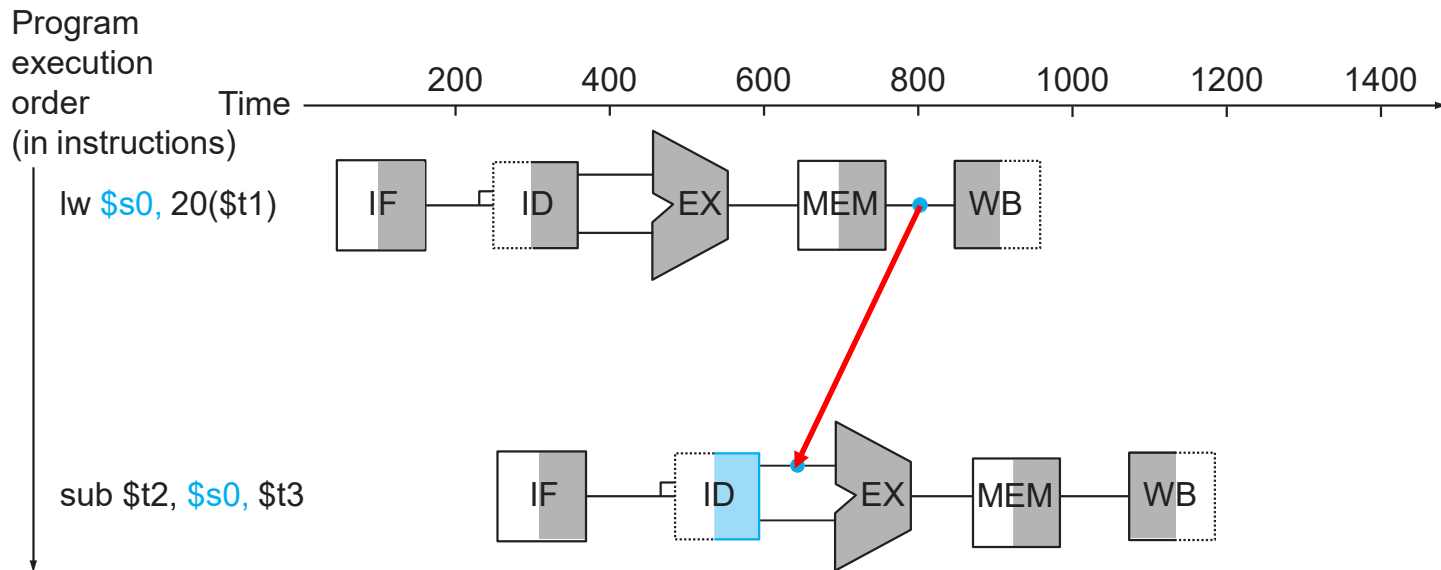
Forwarding Unit in the Pipeline



Forwarding 不能解决的问题

❖ Can't always avoid stalls by forwarding

- If value not ready when needed



Forwarding 不能解决的问题

❖ Can't always avoid stalls by forwarding

- If value not ready when needed
- Solve it by **bubble** in this example

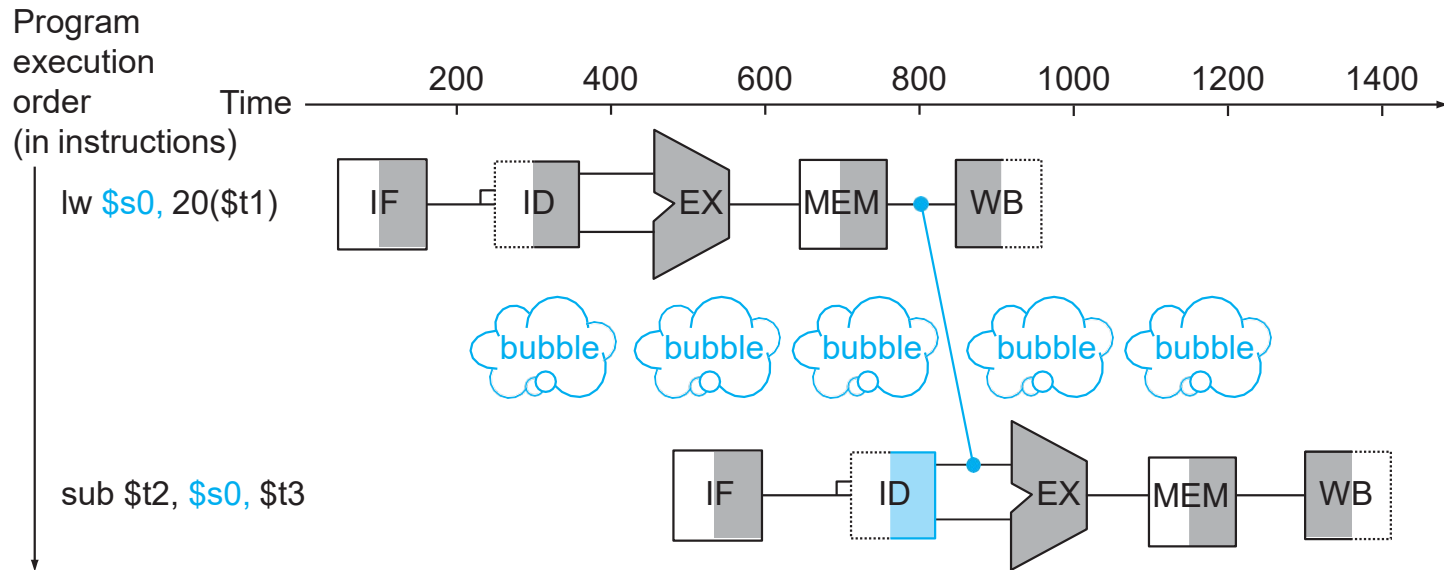


Figure C.10

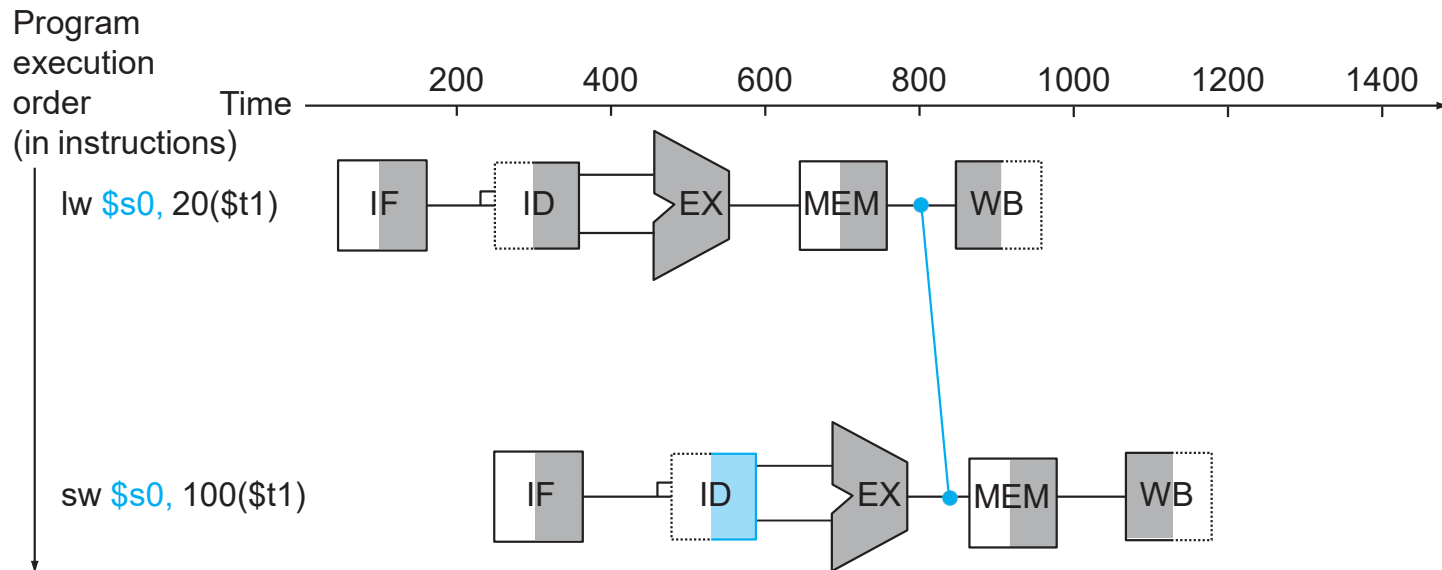
ld	r1, 0(r2)	IF	ID	EX	MEM	WB			
dsub	r4, r1, r5		IF	ID	EX	MEM	WB		
and	r6, r1, r7			IF	ID	EX	MEM	WB	
or	r8, r1, r9				IF	ID	EX	MEM	WB
ld	r1, 0(r2)	IF	ID	EX	MEM	WB			
dsub	r4, r1, r5		IF	ID	stall	EX	MEM	WB	
and	r6, r1, r7			IF	stall	ID	EX	MEM	WB
or	r8, r1, r9				stall	IF	ID	EX	MEM WB

Figure C.10 In the top half, we can see why a stall is needed: The MEM cycle of the load produces a value that is needed in the EX cycle of the `dsub`, which occurs at the same time. This problem is solved by inserting a stall, as shown in the bottom half.

在上半部分，我们可以看出为什么需要停顿，ld指令的MEM周期生成一个值，dsub的EX周期会需要它，而它们是同时发生。通过插入停顿可以解决这一部分，如下半部分所示。

Another forward unit

❖ Add another forward unit in this example



使用这种方法需要增加额外的硬件

Figure C.8

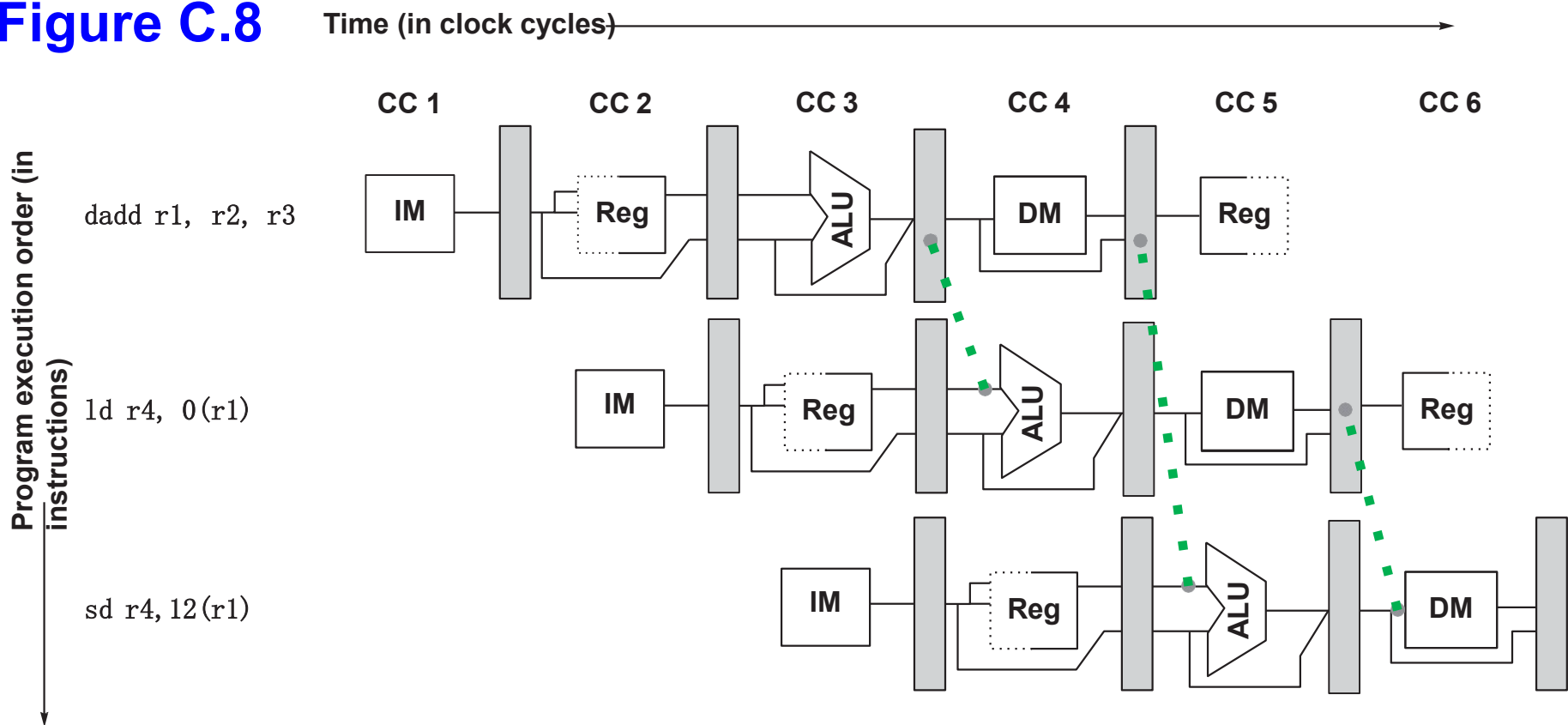
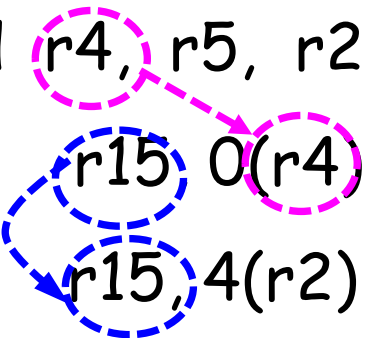


Figure C.8 Forwarding of operand required by stores during MEM. The result of the load is forwarded from the memory output to the memory input to be stored. In addition, the ALU output is forwarded to the ALU input for the address calculation of both the load and the store (this is no different than forwarding to another ALU operation). If the store depended on an immediately preceding ALU operation (not shown above), the result would need to be forwarded to prevent a stall.

在MEM期间执行的写存储操作需要转发操作数。载入结果由存储器输出转发到要存储的存储器输入端。此外，ALU指令被转发到ALU输入，供载入和存储指令进行地址计算（这与转发到另一个ALU操作没有区别）。如果存储操作依赖与其直接相邻的前一个ALU操作（图中未示出），则需要转发其结果，以防止出现停顿。

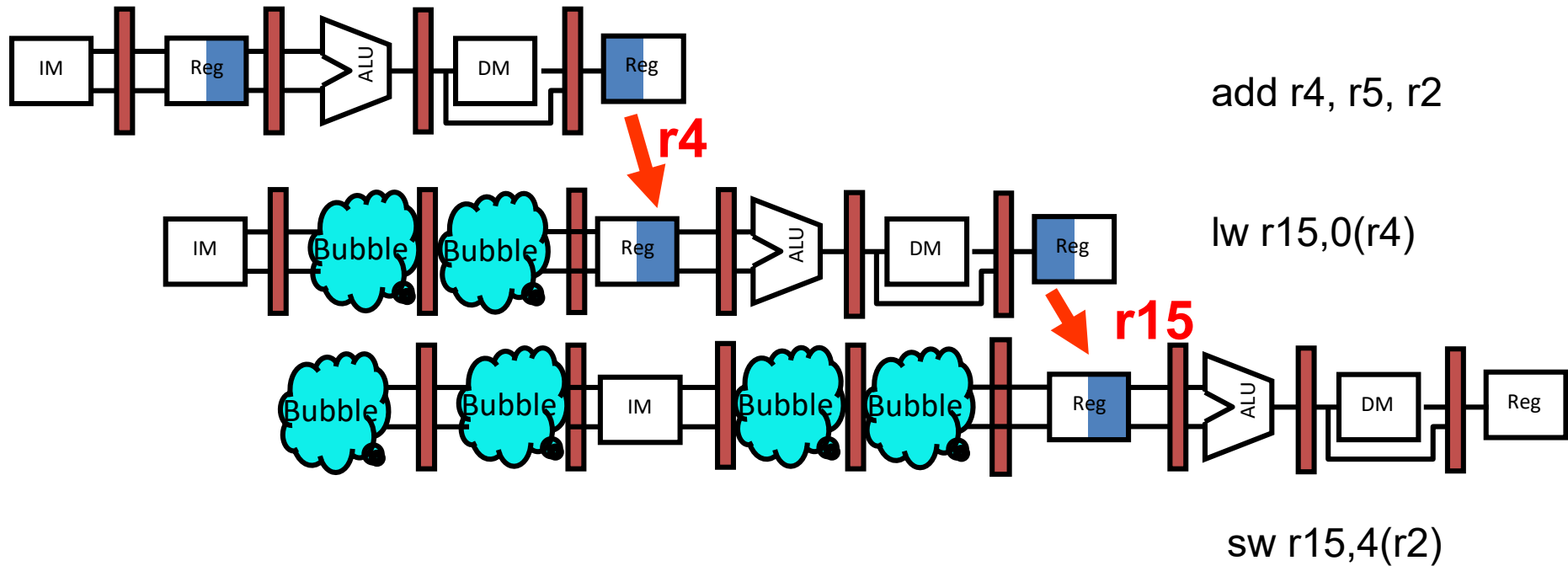
无Forwarding 与有Forwarding比较

❖ Why forwarding?

- add r4, r5, r2
 - lw r15, 0(r4)
 - sw r15, 4(r2)
- 

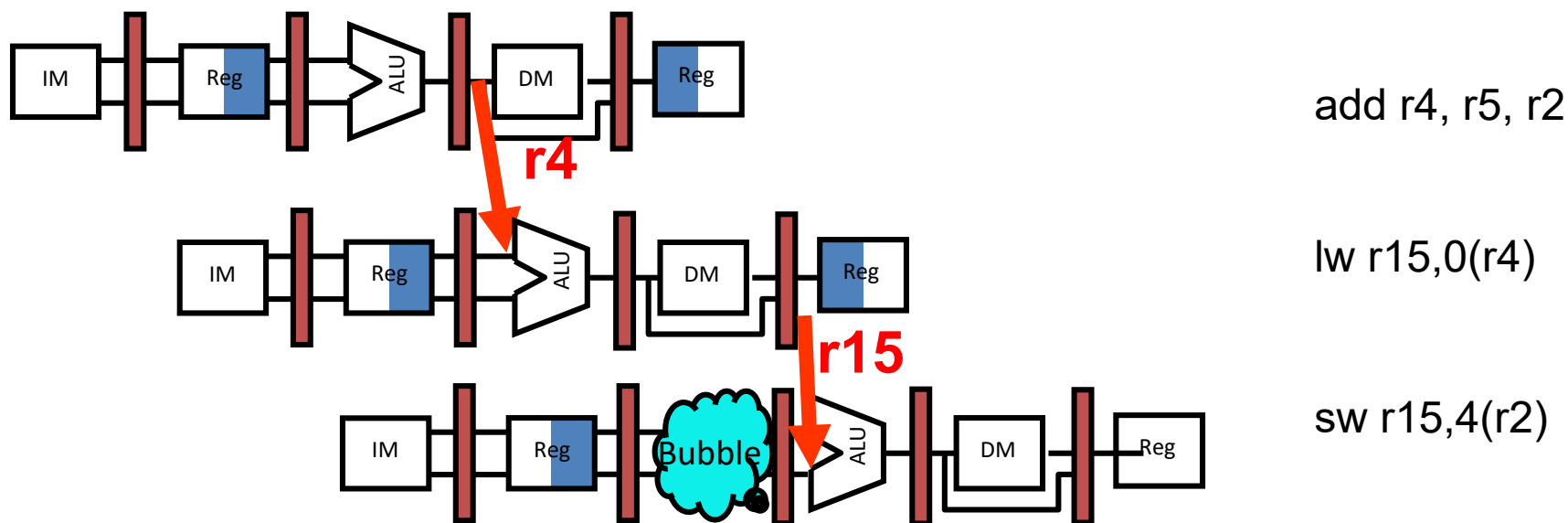
没有forwarding

Time (clock cycles) →



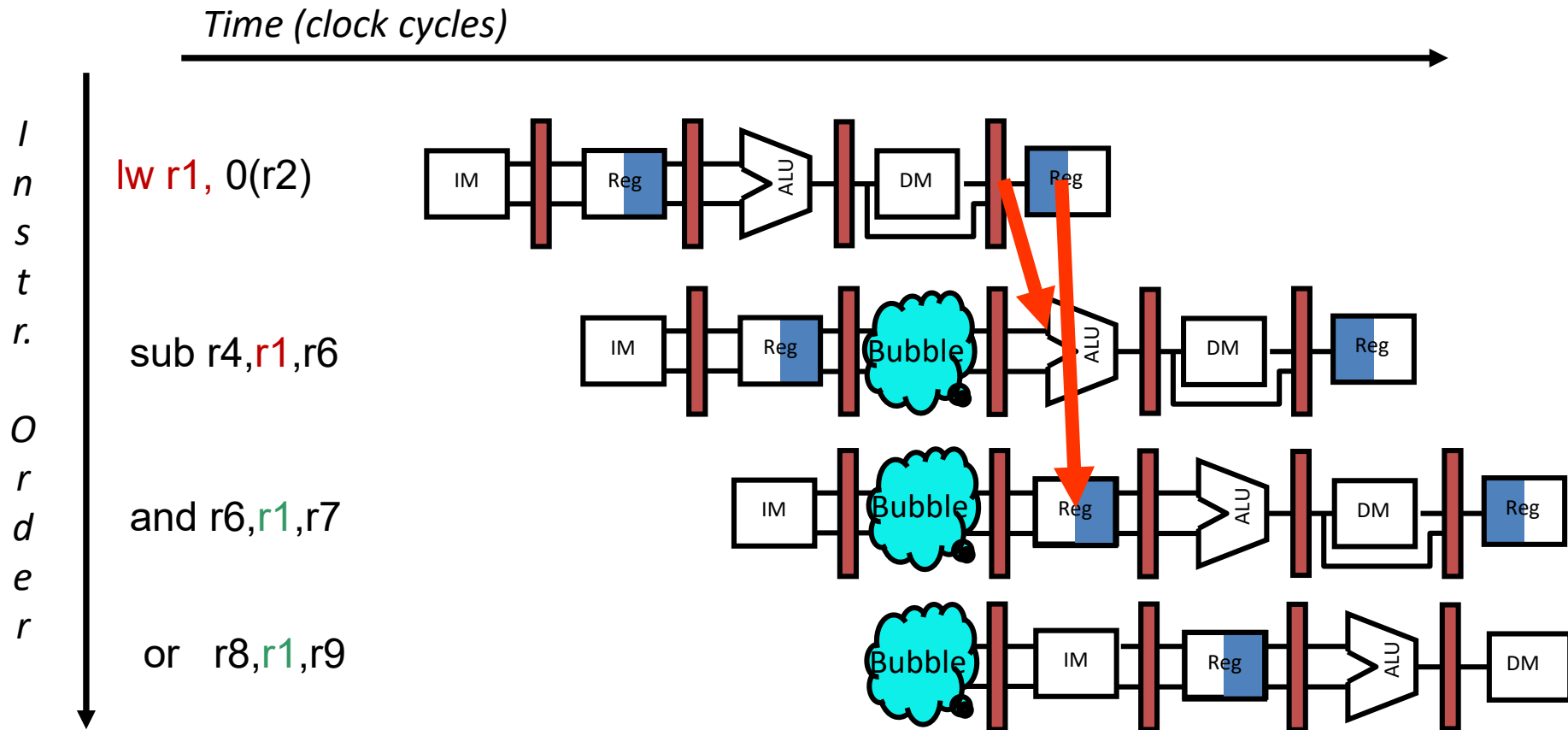
❖ 执行完这3条指令需要11个周期

解决方法: forwarding to ALU



❖ 执行完这3条指令需要8个周期

Load stall的性能影响



load stall的性能影响

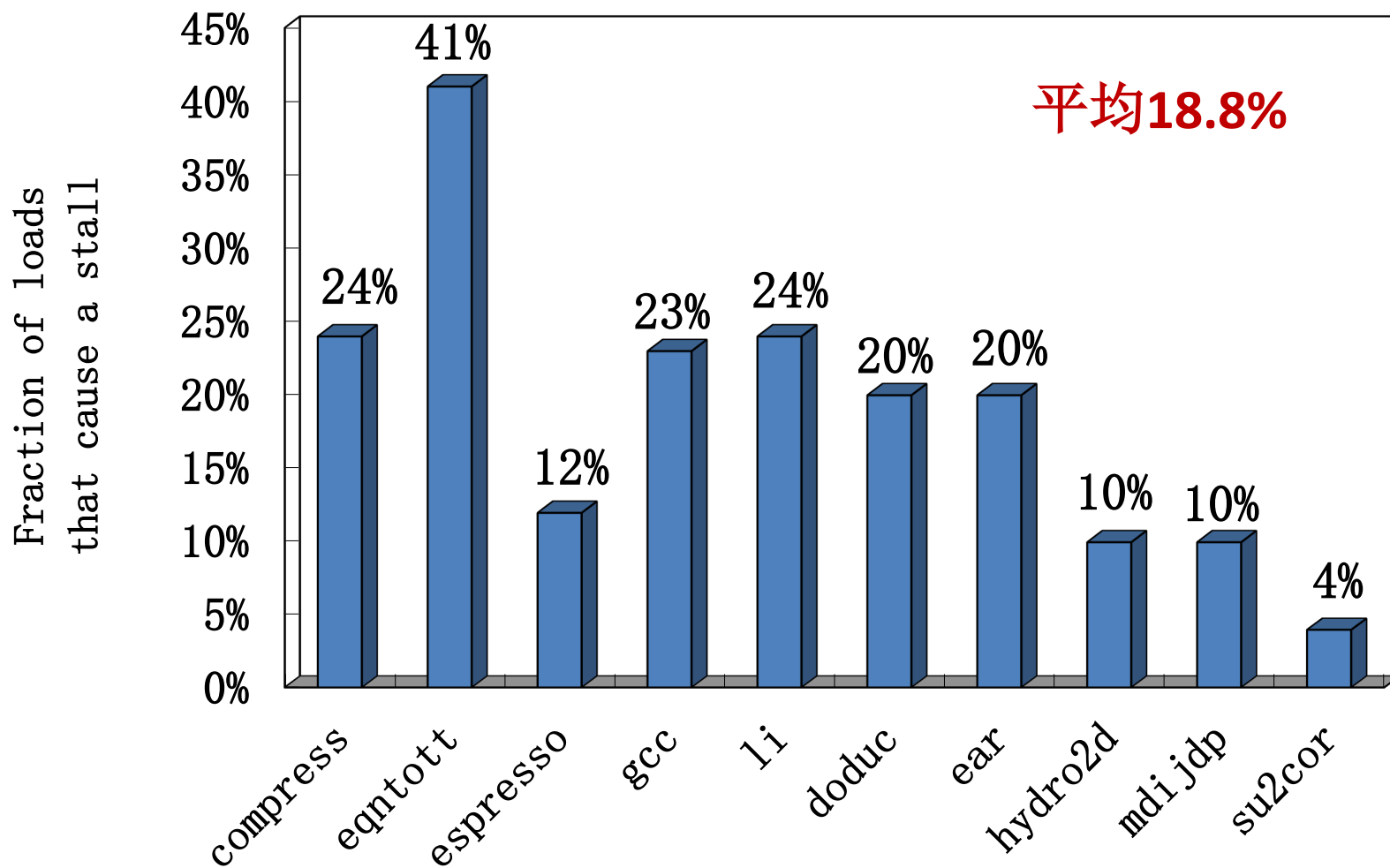
❖ 例子

- 假定程序中**30%** 是**load**指令。
- **有一半的时间**，**load**指令其后的指令依赖**load**的结果。
- 如果冒险引起**一个时钟周期**暂停，相比理想流水线有停顿的流水线性能会降低多少？

❖ Answer

- $CPI = 1 + 30\% \times 50\% \times 1 = 1.15$
- 由于**load**停顿，流水线性能降低 **15%**

不同测试程序中load产生stall的比例



编译器重排序避免load stall

- 为以下表达式产生更快的代码：

$a = b + c;$

$d = e - f;$

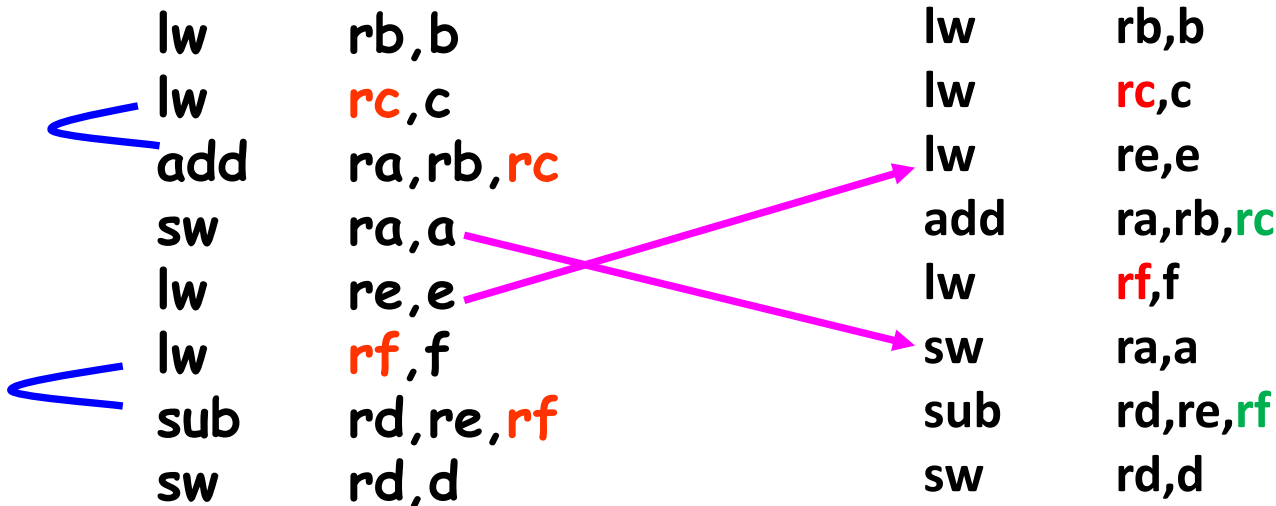
假设 a, b, c, d, e, f 存放在主存中。

- 慢的 code:

```
lw    rb,b
      lw    rc,c
      add   ra,rb,rc
      sw    ra,a
      lw    re,e
      lw    rf,f
      sub   rd,re,rf
      sw    rd,d
```

快的 code:

```
lw    rb,b
      lw    rc,c
      lw    re,e
      add   ra,rb,rc
      lw    rf,f
      sw    ra,a
      sub   rd,re,rf
      sw    rd,d
```



数据冒险总结

❖ 冒险分类

■ 结构冒险

- 硬件资源冲突
- 解决：可以增加硬件资源；或者功能部件完全流水；否则，只有停顿流水线

■ 数据冒险

- 几条指令重叠执行时，一条指令依赖前面指令的结果却没有准备好（还没有计算或存储）
- 解决：Forwarding 通路，编译器重排序；否则就必须停顿

■ 控制冒险

- 在进入ID段时，转移条件和转移目标地址，不能按时提供给IF段取指令

Classroom Test

- ❖ (1) 假定程序中20% 是load指令。有1/4的代码，load后面的指令依赖load的结果。如果冒险引起一个时钟周期暂停，相比理想流水线有停顿的流水线性能会降低多少？
- ❖ (2) 下面是一段C语言代码：
 - ❖ `a[3] = a[0] + a[1];`
 - ❖ `a[4] = a[0] + a[2];`
- ❖ 假设32位整数数组a的开始地址存放在\$t0寄存器中，最初生成的MIPS代码为：
 - ❖ Line1: `lw $t1, 0($t0)`
 - ❖ Line2: `lw $t2, 4($t0)`
 - ❖ Line3: `add $t3, $t1, $t2`
 - ❖ Line4: `sw $t3, 12($t0)`
 - ❖ Line5: `lw $t4, 8($t0)`
 - ❖ Line6: `add $t5, $t1, $t4`
 - ❖ Line7: `sw $t5, 16($t0)`
- ❖ 请问将第5行的`lw $t4, 8($t0)`，移动到第几行的前面可以消除Load暂停。

结构和数据冒险总结

❖ 冒险分类

■ 结构冒险

- 硬件资源冲突
- 解决：可以增加硬件资源；或者功能部件完全流水；否则，只有停顿流水线

■ 数据冒险

- 几条指令重叠执行时，一条指令依赖前面指令的结果却没有准备好（还没有计算或存储）
- 解决：Forwarding 通路，编译器重排序；否则就必须停顿

C.2 流水线的主要障碍—流水线冒险

❖ 一、冒险分类与有停顿流水线性能

❖ 二、结构冒险

❖ 三、数据冒险

❖ 四、控制冒险

四、控制冒险

- ❖ **控制冒险，控制相关：**是指由分支指令引起的相关。它需要根据分支指令的执行结果来确定后面该执行哪个分支上的指令。
- ❖ **怎样解决控制冒险**

流水线冒险-控制冒险

❖ 冒险分类

▪ 结构冒险

- 硬件资源冲突
- 解决：可以增加硬件资源；或者功能部件完全流水；否则，只有停顿流水线

▪ 数据冒险

- 几条指令重叠执行时，一条指令依赖前面指令的结果却没有准备好（还没有计算或存储）
- 解决：Forwarding 通路，编译器重排序；否则就必须停顿

▪ 控制冒险 Control hazards

流水线上执行转移指令：

- 在进入ID段时，转移条件和转移目标地址，不能按时提供给IF段取指令。

控制冒险

❖ 原因

- 转移指令取指令后进入下一个时钟周期时，**转移条件**和**转移目标地址**不能按时提供给IF段取下一条指令。
- 计算**转移目标地址**要花时间
- 对于条件转移，**转移条件**分析要花时间计算

❖ **控制冒险引起MIPS流水线的性能损失比数据冒险大得多。**

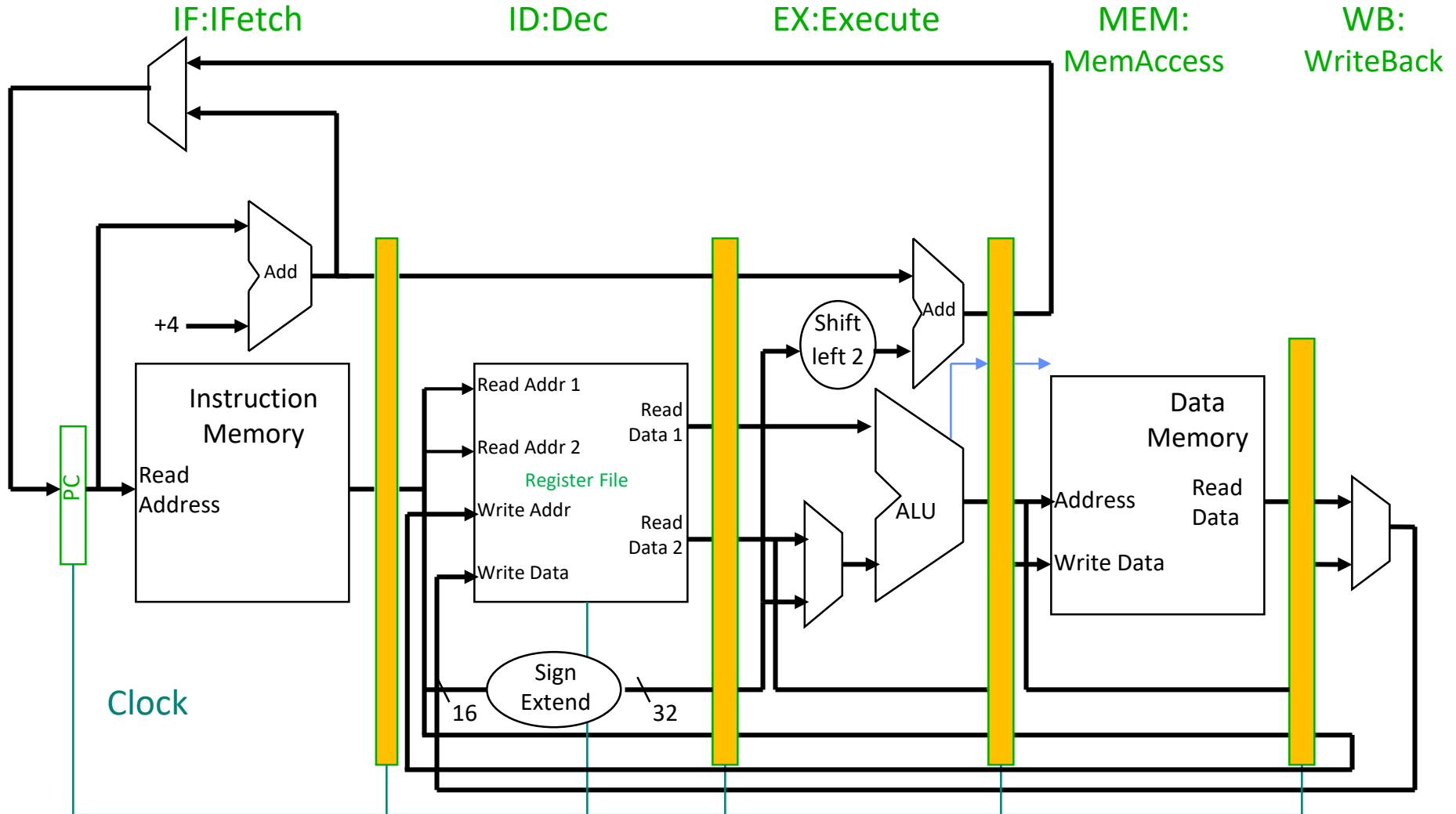
例子: Branches

Address	Instruction	
36	nop	
40	add r30, r30, r30	
44	beqz r1, 28	r1==0 will branches to address 72
48	and r12, r2, r5	We execute all these if r1 !=0
52	or r13, r6, r2	
56	add r14, r2, r2	
60	...	
64	...	
68	...	We execute just these if r1 == 0
72	lw r4, 50(r7)	
76		

Flow of instructions if branch is **taken**: 36, 40, 44, **72**, ...


Flow of instructions if branch is **not taken**: 36, 40, 44, **48**, ...

回忆：基本流水线通路



控制冒险

Address		1	2	3	4	5	6	7	8	9
44	beqz r1, 28	IM	REG	ALU	DM	REG				
48	and r12, r2, r5		IM	REG	ALU	DM	REG			
52	or r13, r6, r2			IM	REG	ALU	DM	REG		
56	add r14, r2, r2				IM	REG	ALU	DM	REG	
60 or 72						IM	REG	ALU	DM	REG



Flow of instructions if branch is **taken**: 44, **72**, ...

Flow of instructions if branch is **not taken**: 44, **48, 52, 56, 60**, ...

处理控制冒险

❖ 4种简单解决方法

- 冻结或冲刷流水线（停顿）
- 预测转移不发生Predict-not-taken (Predict-untaken)
 - 处理每条转移指令都当作未发生转移
- 预测转移发生Predict-taken
 - 处理每条转移指令都当作转移发生
- 转移延迟


❖ 注意：

- 以上任何一种方法都会使**硬件固定**
- 后3种方法，编译时会根据**硬件机制**和**转移行为**对代码进行调度，以获取最佳性能

插入stalls解决冒险

假设在**Data MEM**阶段完成转移条件和转移地址的计算。

Address		1	2	3	4	5	6	7	8	9
44	beqz r1, 28	IM	REG	ALU	DM	REG				
44	stall		IM							
44	stall			IM						
44	stall				IM					
48 or 72						IM	REG	ALU	DM	REG



Flow of instructions if branch is taken: 44, 72, ...

Flow of instructions if branch is not taken: 44, 48, ...

冻结或冲刷流水线

❖ 硬件实现最简单：

- 在转移目标地址确定前，**暂停流水线**或者**废除**已进入流水线但不需要的指令。
- 性能损失是固定的，不能通过软件来减少。

转移Stalls会造成大的性能损失

❖ Problem:

- 程序中有**30%**的转移频率，**理想CPI为1**，插入**stalls**后的性能是多少？假设在Data MEM阶段完成转移条件和转移地址的计算。


❖ Answer:

- $CPI = 1 + 30\% \times 3 = 1.9$
- 这种方法得到的性能大约只有理想性能的一半。

假设转移不发生会怎样？

假设在**Data MEM**阶段完成转移条件和转移地址的计算。

Address		1	2	3	4	5	6	7	8	9
44	beqz r1, 28	IM	REG	ALU	DM	REG				
48	and r12, r2, r5		IM	REG	ALU	DM	REG			
52	or r13, r6, r2			IM	REG	ALU	DM	REG		
56	add r14, r2, r2				IM	REG	ALU	DM	REG	
60						IM	REG	ALU	DM	REG



Flow of instructions if branch is taken: 44, 72, ...

Flow of instructions if branch is **not taken**: 44, 48, 52, 56, **60** ...

预测转移未选中 (Predict –not-taken)

❖ 硬件实现：

- 对每条转移指令都当作**转移未选中**处理 (or as the formal instruction)
 - 当转移未选中时，取到的指令正好是要继续执行的，**没有任何停顿**。
 - 如果转移选中，则重新取转移目标指令，此时引起**3**个指令被废弃。(should turn the fetched instruction into a no-op)


❖ 编译器：

- 编译时，将使用频率最高的代码放在转移未选中路径上，改进性能。

如果转移是发生的会怎样？

假设在**Data MEM**阶段完成转移条件和转移地址的计算。

Address		1	2	3	4	5	6	7	8	9
44	beqz r1, 28	IM	REG	ALU	DM	REG				
48	and r12, r2, r5		IM	REG	nop	nop	nop			
52	or r13, r6, r2			IM	nop	nop	nop	nop		
56	add r14, r2, r2				nop	nop	nop	nop	nop	
72						IM	REG	ALU	DM	REG

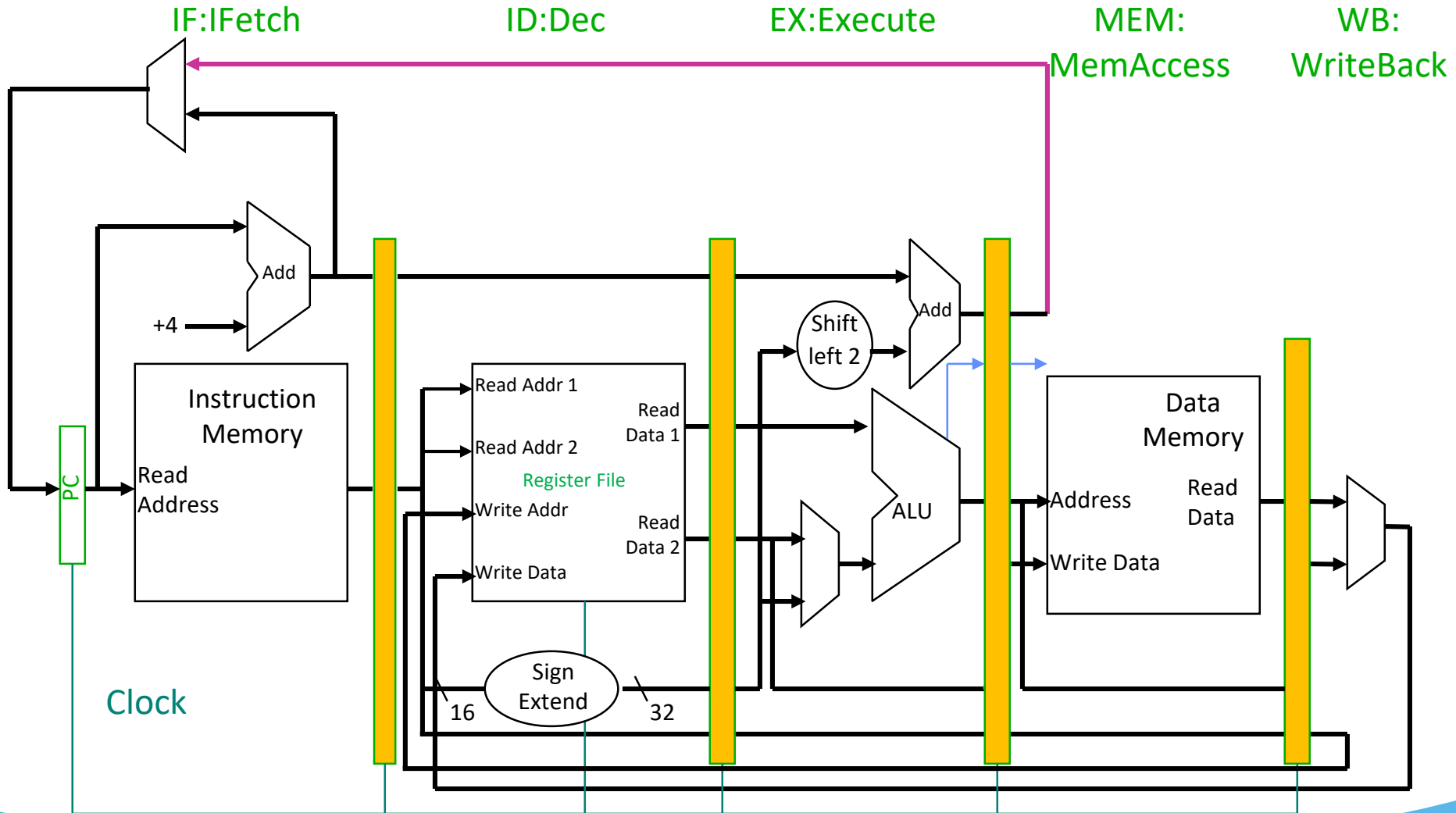


Flow of instructions if branch is taken: 44, 72, ...

Flow of instructions if branch is not taken: 44, 48, ...

5级流水线

❖ 假设在Data MEM阶段完成转移条件和转移地址的计算。



预测转移选中 (Predict –taken)

❖ 硬件实现

- 将所有转移指令都当作转移选中处理 (evidence: more than 60% braches are taken)
- 只要转移目标地址计算出来，就开始取目标指令。
- 只有在目标地址计算比转移条件更早产生才有用。
- 对于经典5段流水线（如前页图），是在DM阶段得到转移目标地址，因此转移选中方法没有任何益处。

❖ 编译器

- 编译时，将使用频率最高的代码放在转移选中路径上，改进性能。

预测转移选中的流水线状态

❖ 本页PPT：目标地址在ID级确定，转移条件在MEM级确定

Branch is **taken**: 1 stall

44 beqz r1, 28	IF	ID	EX	MEM	WB		
48 and r12, r2, r5		IF	idle	idle	idle	idle	
72 lw r4, 50(r7)			IF	ID	EX	MEM	WB
76				IF	ID	EX	MEM
80					IF	ID	EX

not taken 3 stall

44 beqz r1, 28	IF	ID	EX	MEM	WB		
48 and r12, r2, r5		IF	idle	idle	idle	idle	
72 lw r4, 50(r7)			IF	idle	idle	idle	idle
76				IF	idle	idle	idle
48 and r12, r2, r5					IF	ID	EX

已经取到48号地址指令，如果转移最终没有发生，有必要第2次取该指令吗？目前这种情况下是重新取，所以停顿3个周期

预测转移选中的流水线状态

❖ 本页PPT：目标地址在ID级确定，转移条件也在ID级确定

Branch is **taken**: 1 stall

44 beqz r1, 28	IF	ID	EX	MEM	WB		
48 and r12, r2, r5		IF	idle	idle	idle	idle	
72 lw r4, 50(r7)			IF	ID	EX	MEM	WB
76				IF	ID	EX	MEM
80					IF	ID	EX

not taken 1 stall

44 beqz r1, 28	IF	ID	EX	MEM	WB		
48 and r12, r2, r5		IF	idle	idle	idle	idle	
48 and r12, r2, r5			IF	ID	EX	MEM	WB

已经取到48号地址指令，如果转移最终没有发生，有必要第2次取该指令吗？目前这种情况下是重新取，所以停顿1个周期

提早后结果：MIPS改善后的数据通路

- 在BEQZ指令后，只需要**1**个额外周期就可以知道正确地址
- 在MIPS，条件转移指令其后的这个周期称为**转移延迟槽 the branch delay slot**

预测转移未选中/选中的问题

❖ 流水线按照假定的转移方向执行

- 对于选择方向是正确的，可以节省时钟周期

❖ 假定的转移方向错误

- 开始执行的是错误指令
- 要修复，则必须确认这些错误指令没有真正执行
- 尤其是，必须保证错误指令没有改变机器状态

预测转移未选中predict-not-taken（硬件固定）

❖ 本页PPT：目标地址在ID级确定，转移条件也是在ID级确定

转移未选中： No stall

40 add r30,r30,r30	IF	ID	EX	MEM	WB		
44 beqz r1, 24		IF	ID	EX	MEM	WB	
48 and r12, r2, r5			IF	ID	EX	MEM	WB
52 or r13, r6, r2				IF	ID	EX	MEM

转移发生： 1 stall

40 add r30,r30,r30	IF	ID	EX	MEM	WB		
44 beqz r1, 24		IF	ID	EX	MEM	WB	
48 and r12, r2, r5			IF	idle	idle	idle	idle
72 lw r4, 50(r7)				IF	ID	EX	MEM
76					IF	ID	EX

转移延迟 (Delayed branch)

❖ 对于转移条件与地址计算在ID级的流水线

❖ 好的方面

- 只需要1个周期计算出正确的转移地址和条件
- 因此，不需要2或3个周期的nop或stall

❖ 不如意的方面

- 总是有1cycle，总是要等待（指当没有采用减少停顿的措施时）
- 在MIPS中，无论转移是否发生，这个周期的指令总是要进入流水线执行 (hardware scheme)

注：延迟槽不利于软件调试，RISC-V不使用它。

如何利用转移延迟槽

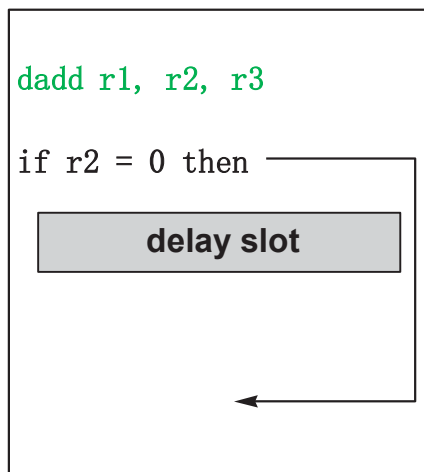
❖ branch delay slot

- 转移指令后的指令周期被用于转移地址计算，至少需要1个周期的延迟。
- 因此，可以将延迟槽作为自由的指令周期加以利用。

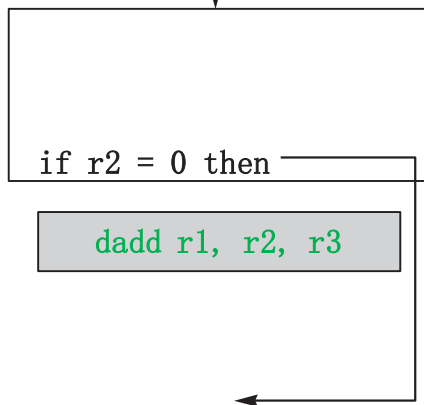
❖ Consequence

- 编译器可以将合适的代码调整到这个“slot”中，如果只放入nop就浪费了。(compiler scheme)。

怎样调整代码？

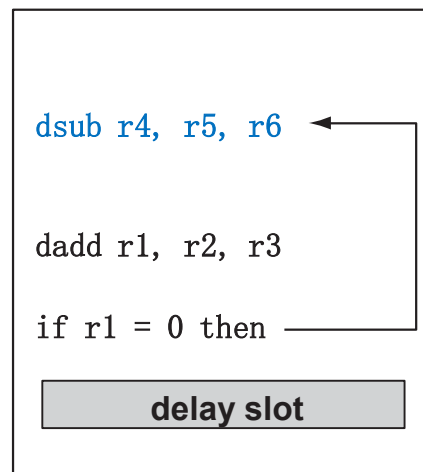


becomes

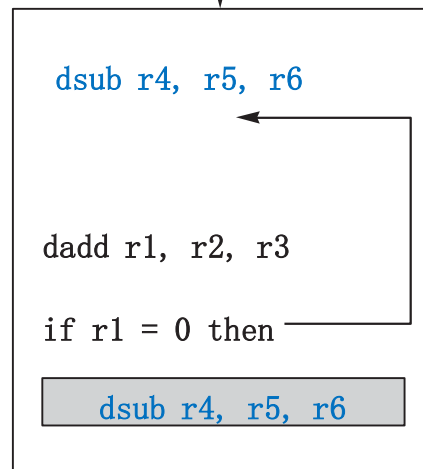


(a) From before

如果可能，尽量采用这种方法。

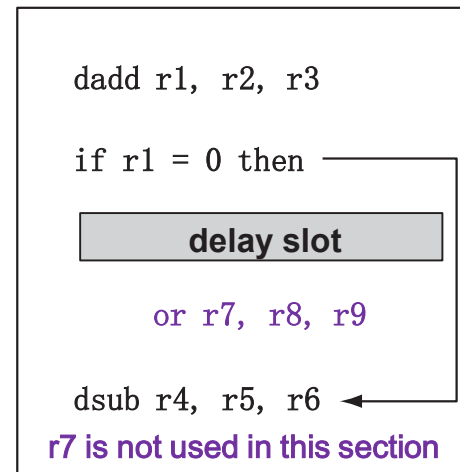


becomes

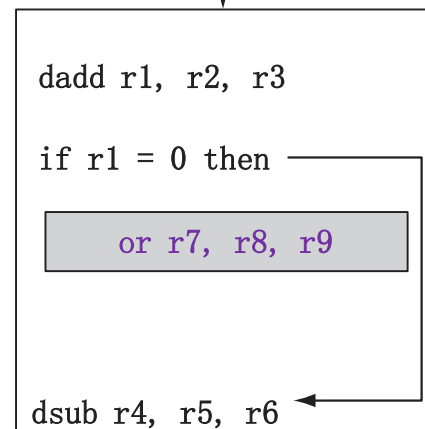


(b) From target

条件的判定需要dadd指令的结果，这种情况下无法使用方法(a)。采用这种方法要求延迟槽在条件不成立时能够取消。



becomes



(c) From fall-through

条件的判定需要dadd指令的结果，这种情况下要求延迟槽的指令在条件不成立时不影响其它代码的正常执行。本例中条件不成立的部分没有使用r7。也可以让延迟槽在条件不成立时能够取消。

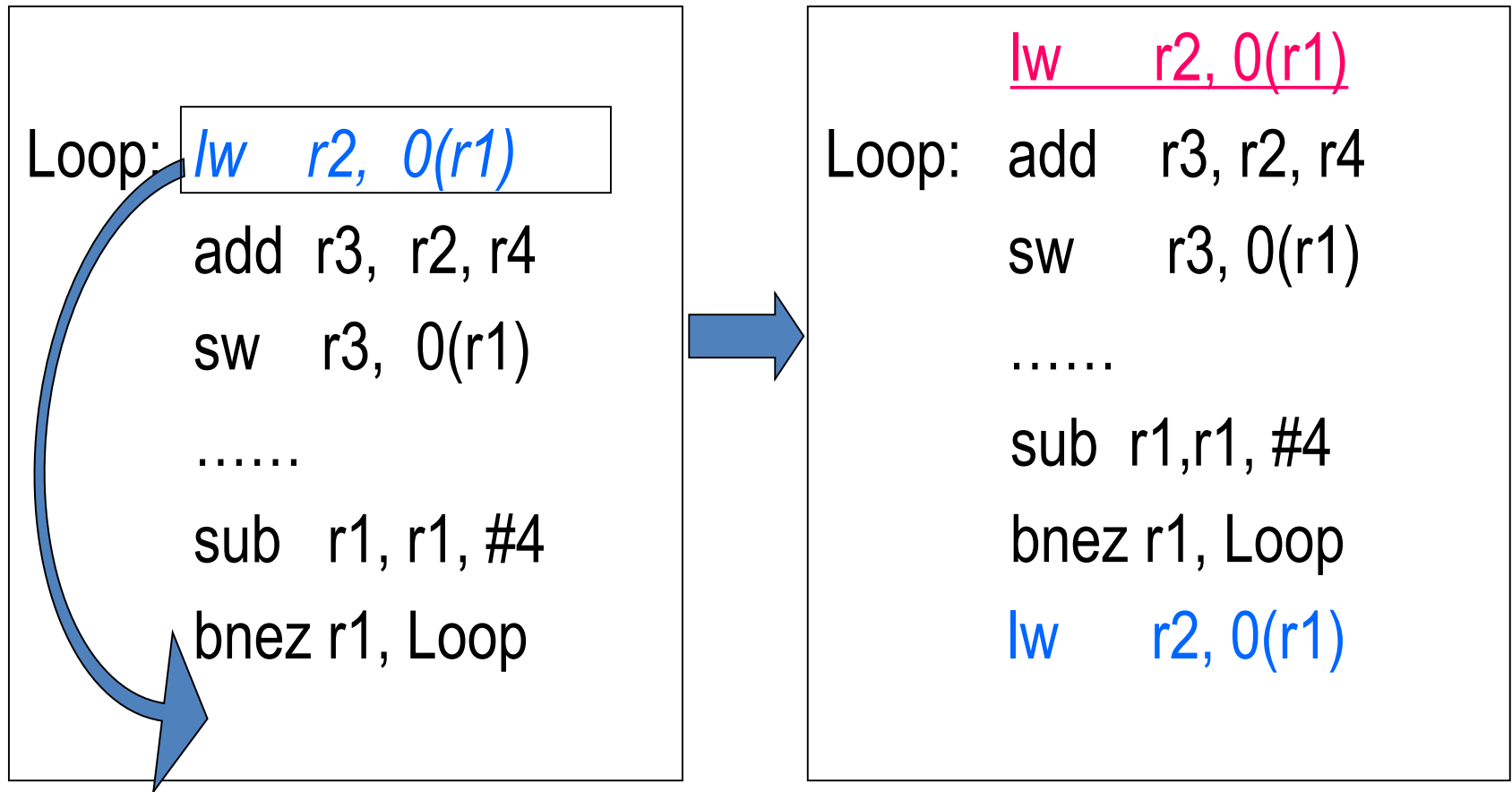
例子：重写代码(方法a)

Address	Instruction	
36	nop	
40	add r30, r30, r30 -> beqz r1, 32	This branches to address 72
44	beqz r1, 28 -> add r30, r30, r30	Delay slot: always executed.
48	and r12, r2, r5	We execute all these if r1 != 0
52	or r13, r6, r2	
56	add r14, r2, r2	
60	...	
64	...	
68	...	
72	lw r4, 50(r7)	We execute just these if r1 == 0
76		

(a) From before

如果可能，尽量采用这种方法。本例中，地址**44**的指令不依赖地址**40**的指令，所以将**add r30, r30, r30**放入延迟槽。

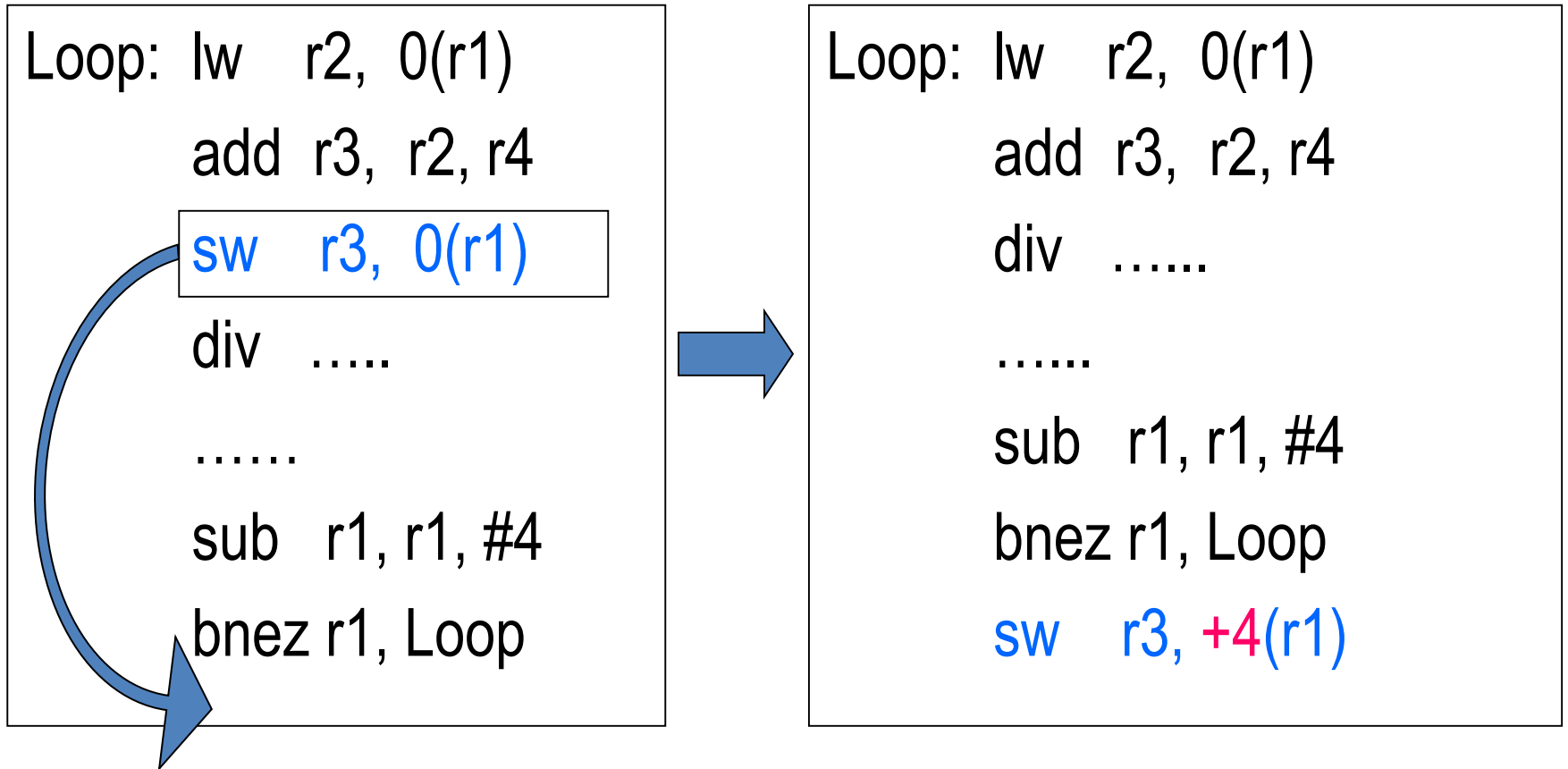
例子：重写代码(方法b)



(b) From target

条件的判定需要**sub**指令的结果，这种情况下无法将**sub**指令放入延迟槽。采用这种方法要求延迟槽在条件不成立时能够取消。

例子：重写代码(c)



(c) From fall-through

条件的判定需要**sub**指令的结果，这种情况下要求延迟槽的指令在条件不成立时不影响其它代码的正常执行。本例中需要改写左边的**sw r3,0(r1)**语句，而且要求延迟槽在条件不成立时能够取消。

转移延迟的限制

- ❖ 调度到延迟槽的指令有限制。
- ❖ 编译器预测转移是否发生的准确率，决定了调度后程序执行性能提高的程度。

撤销功能

- ❖ 考虑转移预测是预测转移发生还是不发生。
- ❖ 如果转移预测是错误的，CPU能够将转移延迟槽中的指令转换为一条空操作指令。
- ❖ 能够减少编译器选择有用指令进入转移延迟槽的复杂性。

带有撤销功能的转移预测

(预测转移不发生)

Untaken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	ID	EX	MEM	WB			
Instruction $i + 2$			IF	ID	EX	MEM	WB		
Instruction $i + 3$				IF	ID	EX	MEM	WB	
Instruction $i + 4$					IF	ID	EX	MEM	WB
Taken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	idle	idle	idle	idle			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB

Figure C.12 The predicted-not-taken scheme and the pipeline sequence when the branch is untaken (top) and taken (bottom). When the branch is untaken, determined during ID, we fetch the fall-through and just continue. If the branch is taken during ID, we restart the fetch at the branch target. This causes all instructions following the branch to stall 1 clock cycle.

分支未选中（上）和被选中（下）时的预测未选中机制和流水线序列

当分支未被选中时（在ID期间确定），我们提取未选中指令，继续进行。当在ID期间确定选中该分支时，则在分支目标处重新开始提取。这将导致该分支后面的所有指令停顿1个时钟周期。

Figure C.13

Untaken branch instruction	IF	ID	EX	MEM	WB				
Branch delay instruction ($i + 1$)		IF	ID	EX	MEM	WB			
Instruction $i + 2$			IF	ID	EX	MEM	WB		
Instruction $i + 3$				IF	ID	EX	MEM	WB	
Instruction $i + 4$					IF	ID	EX	MEM	WB
Taken branch instruction	IF	ID	EX	MEM	WB				
Branch delay instruction ($i + 1$)		IF	ID	EX	MEM	WB			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB

Figure C.13 The behavior of a delayed branch is the same whether or not the branch is taken. The instructions in the delay slot (there is only one delay slot for MIPS) are executed. If the branch is untaken, execution continues with the instruction after the branch delay instruction; if the branch is taken, execution continues at the branch target. When the instruction in the branch delay slot is also a branch, the meaning is unclear: If the branch is not taken, what should happen to the branch in the branch delay slot? Because of this confusion, architectures with delay branches often disallow putting a branch in the delay slot.

无论分支是否选中，延迟分支的行为特性都是相同的

延迟槽中的指令（对于MIPS只有一个延迟槽）被执行。如果分支未被选中，则继续执行分支延迟指令之后的指令；如果分支被选中，则继续在分支目标处执行。当分支延迟槽中的指令也是分支时，其含义就有些模糊：如果该分支未被选中，延迟分支槽中的分支应当怎么办呢？由于这一混淆，**采用延迟分支的体系结构经常禁止在延迟槽中放入分支。**

关于转移延迟

- ❖ 转移延迟在部分RISC处理器中采用。
- ❖ 通常，转移延迟长度大于1。然而，由于编译器的复杂性，总是使用 **1** 个延迟槽。

例子：三种解决控制冒险方案的性能比较

❖ MIPS R4000，更深的流水线（8级）

- 转移目标地址计算，花费三个流水段。
- 转移条件在EX段形成，需要四个流水段。
- 假定转移频率如下：
 - Unconditional branch 4%
 - Conditional branch, untaken 6%
 - Conditional branch, taken 10%

❖ 3种方案：

- Stall pipeline
- Predict taken
- Predict untaken

s: stall

目标地址计算：花费三个流水段。
条件在EX段形成，需要四个流水段。

3种情况的流水线状态

■ **Stall uncond.** L1 L2 L3 L4 L5 L6 2 stall

s **s** L1(branch target)

taken/untaken: L1 L2 L3 L4 L5 L6 3 stall

s **s** **s** L1(branch target/i+1)

■ Predict taken:

L1 L2 L3 L4 L5 L6

uncond. / taken: **s** **s** L1(branch target) 2 stall

untaken: s s L1 L1(i+1) 3 stall

■ **Predict untaken:**

L1 L2 L3 L4 L5 L6

untaken: L1 L2 L3 L4 L5 0 stall

uncond. s s L1 L2 L3 2 stall

taken: s s L1 L1(branch target) 3 stall

3种方案Stall的比较结果

Branch scheme	Additiona to the CPI						All branches (20%)
	Unconditional Branches(4%)		Taken cond. Branches(10%)		Untaken cond. Branches(6%)		
Stall pipeline	2	0.08	3	0.30	3	0.18	0.56
Predict taken	2	0.08	2	0.20	3	0.18	0.46
Predict untaken	2	0.08	3	0.30	0	0.00	0.38

Example

For a deeper pipeline, such as that in a MIPS R4000, it takes at least three pipeline stages before the branch-target address is known and an additional cycle before the branch condition is evaluated, assuming no stalls on the registers in the conditional comparison. A three-stage delay leads to the branch penalties for the three simplest prediction schemes listed in Figure C.15. 对于一个更深的流水线，比如在MIPSR4000中，在知道分支目标地址之前至少需要三个流水级，在计算分支条件之前需要增加一个周期，这里假定条件比较时寄存器中没有停顿。三级延迟导致表C-7中所列三种最简单预测机制的分支代价。

Find the effective addition to the CPI arising from branches for this pipeline, assuming the following frequencies: 假定有如下概率，计算因分支使该流水线的CPI增加了多少？

Unconditional branch	4%
Conditional branch, untaken	6%
Conditional branch, taken	10%

Branch scheme	Penalty unconditional	Penalty untaken	Penalty taken
Flush pipeline	2	3	3
Predicted taken	2	3	2
Predicted untaken	2	0	3

Figure C.15 Branch penalties for the three simplest prediction schemes for a deeper pipeline.

Unconditional branch	4%
Conditional branch, untaken	6%
Conditional branch, taken	10%

Answer

Branch scheme	Penalty unconditional	Penalty untaken	Penalty taken
Flush pipeline	2	3	3
Predicted taken	2	3	2
Predicted untaken	2	0	3

Figure C.15 Branch penalties for the three simplest prediction schemes for a deeper pipeline.

Additions to the CPI from branch costs				
Branch scheme	Unconditional branches	Untaken conditional branches	Taken conditional branches	All branches
Frequency of event	4%	6%	10%	20%
Stall pipeline	$4\% \times 2 = 0.08$	$6\% \times 3 = 0.18$	$10\% \times 3 = 0.30$	0.56
Predicted taken	$4\% \times 2 = 0.08$	$6\% \times 3 = 0.18$	$10\% \times 2 = 0.20$	0.46
Predicted untaken	$4\% \times 2 = 0.08$	$6\% \times 0 = 0.00$	$10\% \times 3 = 0.30$	0.38

Figure C.16 CPI penalties for three branch-prediction schemes and a deeper pipeline.

The differences among the schemes are substantially increased with this longer delay. If the base CPI were 1 and branches were the only source of stalls, the ideal pipeline would be $(1+0.56)/1=1.56$ times faster than a pipeline that used the stall-pipeline scheme. The predicted-untaken scheme would be $(1+0.56)/(1+0.38)=1.13$ times better than the stall-pipeline scheme under the same assumptions.

流水线冒险

❖ 冒险分类

■ 结构冒险

- 硬件资源冲突.

■ 数据冒险

- 几条指令重叠执行时，一条指令依赖前面指令的结果却没有准备好（还没有计算或存储）

■ 控制冒险 Control hazards

- 转移指令进入ID时，转移条件和转移目标地址是不能按时提供给IF取指令的。
- 解决方法：提早计算目标地址和条件，冻结或冲刷流水线，
预测转移未选中，预测转移选中，转移延迟

Exercise C.1

- ❖ Use the following code fragment:
- ❖ 使用以下代码进行分析:
- ❖ Loop: LD R1,0(R2) ;load R1 from address 0+R2
- ❖ DADDI R1,R1,#1 ;R1=R1+1
- ❖ SD R1,0(R2) ;store R1 at address 0+R2
- ❖ DADDI R2,R2,#4 ;R2=R2+4
- ❖ DSUB R4,R3,R2 ;R4=R3-R2
- ❖ BNEZ R4,Loop ;branch to Loop if R4!=0
- ❖ Assume that the initial value of R3 is R2 + 396.
- ❖ 假定 R3 的初始值为 R2+396 。

Exercise C.1(A)

- ❖ Use the following code fragment:
- ❖ 使用以下代码进行分析:
- ❖ Loop: LD R1,0(R2) ;load R1 from address 0+R2
- ❖ DADDI R1,R1,#1 ;R1=R1+1
- ❖ SD R1,0(R2) ;store R1 at address 0+R2
- ❖ DADDI R2,R2,#4 ;R2=R2+4
- ❖ DSUB R4,R3,R2 ;R4=R3-R2
- ❖ BNEZ R4,Loop ;branch to Loop if R4!=0
- ❖ Assume that the initial value of R3 is R2 + 396.
- ❖ 假定 R3 的初始值为 R2+396 。
- ❖ (a) Data hazards are caused by data dependences in the code. Whether a dependency causes a hazard depends on the machine implementation (i.e., number of pipeline stages). List all of the data dependences in the code above. Record the register, source instruction, and destination instruction; for example, there is a data dependency for register R1 from the LD to the DADDI.
- ❖ 数据冒险是由代码中的数据相关性导致的。相关性是否会导致冒险, 取决于机器实现 (即流水级的数目)。列出上述代码中的所有数据相关。记录寄存器、源指令和目标指令; 例如, 从R1 LD 到 DADDI, 存在对于寄存器 R1 的数据相关性。

Exercise C.1(A)

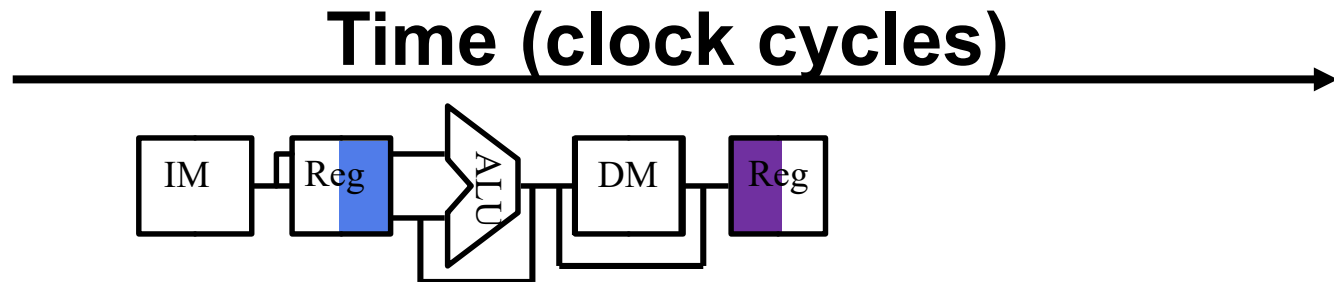
- ❖ Loop: LD R1,0(R2) ;load R1 from address 0+R2
- ❖ DADDI R1,R1,#1 ;R1=R1+1
- ❖ SD R1,0(R2) ;store R1 at address 0+R2
- ❖ DADDI R2,R2,#4 ;R2=R2+4
- ❖ DSUB R4,R3,R2 ;R4=R3-R2
- ❖ BNEZ R4,Loop ;branch to Loop if R4!=0
- ❖ (a)
- ❖ 数据冒险是由代码中的数据相关性导致的。相关性是否会导致冒险, 取决于机器实现 (即流水级的数目)。列出上述代码中的所有数据相关。记录寄存器、源指令和目标指令; 例如, 从R1 LD 到 DADDI, 存在对于寄存器 R1 的数据相关性。
- ❖ R1 LD DADDI
- ❖ R1 DADDI SD
- ❖ R2 DADDI DSUB
- ❖ R2 DADDI LD
- ❖ R2 DADDI SD
- ❖ R4 DSUB BNEZ

Exercise C.1(B)

- ❖ 给出这一指令序列对于 5 级 RISC 流水线的时序, 该流水线没有任何前递或旁路硬件, 但假定在同一时钟周期中的寄存器读取与写入通过寄存器堆就像图 C-6。请使用如图 C-5 中所示的流水线时序表。假定该分支是通过冲刷流水线来处理的。如果所有存储器访问耗时 1 个周期, 这个循环的执行需要多少个周期?

- ❖ Loop: LD R1,0(R2) ;load R1 from address 0+R2
- ❖ DADDI R1,R1,#1 ;R1=R1+1
- ❖ SD R1,0(R2) ;store R1 at address 0+R2
- ❖ DADDI R2,R2,#4 ;R2=R2+4
- ❖ DSUB R4,R3,R2 ;R4=R3-R2
- ❖ BNEZ R4,Loop ;branch to Loop if R4!=0
- ❖ 假定 R3 的初始值为 R2+396 。

Figure C.6



❖ **5级 RISC 流水线**, 没有任何前递或旁路硬件, 假定在同一时钟周期中的寄存器先写入后读取。分支是通过冲刷流水线来处理的。如果所有存储器访问耗时 1 个周期, 这个循环的执行需要多少个周期?

- ❖ **Loop: LD R1,0(R2)** ;load R1 from address 0+R2

◆ **DADDI R1,R1,#1 ;R1=R1+1**

❖ **SD R1,0(R2)** ;store R1 at address 0+R2

❖ **DADDI R2,R2,#4 ;R2=R2+4**

❖ **DSUB R4,R3,R2 ;R4=R3-R2**

◆ **BNEZ R4,Loop** ;branch to Loop if R4!=0

❖ 假定 R3 的初始值为 $R2+396$ 。

Exercise C.1(B)

[illegible]

Exercise C.1(B)

- ❖ Loop: LD R1,0(R2) ;load R1 from address 0+R2
- ❖ DADDI R1,R1,#1 ;R1=R1+1
- ❖ SD R1,0(R2) ;store R1 at address 0+R2
- ❖ DADDI R2,R2,#4 ;R2=R2+4
- ❖ DSUB R4,R3,R2 ;R4=R3-R2
- ❖ BNEZ R4,Loop ;branch to Loop if R4!=0
- ❖ 假定 R3 的初始值为 R2+396 。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LD R1,0(R2)	F	D	X	M	W													
DADDI R1,R1,#1		F	s	s	D	X	M	W										
SD R1,0(R2)					F	s	s	D	X	M	W							
DADDI R2,R2,#4								F	D	X	M	W						
DSUB R4,R3,R2									F	s	s	D	X	M	W			
BNEZ R4,Loop												F	s	s	D	X	M	W
LD R1,0(R2)																	F	D

Since the initial value of R3 is $R2 + 396$ and equal instance of the loop adds 4 to R2, the total number of iterations is 99. Notice that there are 8 cycles lost to RAW hazards including the branch instruction. Two cycles are lost after the branch because of the instruction flushing. It takes 16 cycles between loop instances; the total number of cycles is $98 \times 16 + 18 = 1584$. The last loop takes two addition cycles since this latency cannot be overlapped with additional loop instances.

❖ **5级 RISC 流水线**, 有完整旁路硬件, 假定在同一时钟周期中的寄存器先写入后读取。假定在处理分支时, **预测它未被选中**。如果所有存储器访问耗时 1 个周期, 这个循环的执行需要多少个周期?

❖ **Loop: LD R1,0(R2) ;load R1 from address 0+R2**

❖ **DADDI R1,R1,#1 ;R1=R1+1**

❖ **SD R1,0(R2) ;store R1 at address 0+R2**

❖ **DADDI R2,R2,#4 ;R2=R2+4**

❖ **DSUB R4,R3,R2 ;R4=R3-R2**

❖ **BNEZ R4,Loop ;branch to Loop if R4!=0**

❖ **假定 R3 的初始值为 R2+396 。**

Exercise C.1(C)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LD R1,0(R2)	F	D	X	M	W													
DADDI R1,R1,#1		F	D	s	X	M	W											
SD R1,0(R2)			F	s	D	X	M	W										
DADDI R2,R2,#4					F	D	X	M	W									
DSUB R4,R3,R2						F	D	X	M	W								
BNEZ R4,Loop							F	s	D	X	M	W						
Incorrect instruction									F	s	s	s	s					
LD R1,0(R2)										F	D	X	M	W				

- ❖ **Loop:** LD R1,0(R2) ;load R1 from address 0+R2
- ❖ DADDI R1,R1,#1 ;R1=R1+1
- ❖ SD R1,0(R2) ;store R1 at address 0+R2
- ❖ DADDI R2,R2,#4 ;R2=R2+4
- ❖ DSUB R4,R3,R2 ;R4=R3-R2
- ❖ BNEZ R4,Loop ;branch to Loop if R4!=0
- ❖ 假定 R3 的初始值为 R2+396 。 预测它未被选中

Exercise C.1(C)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LD R1,0(R2)	F	D	X	M	W													
DADDI R1,R1,#1		F	D	s	X	M	W											
SD R1,0(R2)			F	s	D	X	M	W										
DADDI R2,R2,#4					F	D	X	M	W									
DSUB R4,R3,R2						F	D	X	M	W								
BNEZ R4,Loop							F	s	D	X	M	W						
Incorrect instruction									F	s	s	s	s					
LD R1,0(R2)										F	D	X	M	W				

Again we have 99 iterations. There are two RAW stalls and a flush after the branch since the branch is taken. The total number of cycles is $9 * 98 + 12 = 894$. The last loop takes three addition cycles since this latency cannot be overlapped with additional loop instances.

- ❖ 5级 RISC 流水线, 有完整旁路硬件, 假定在同一时钟周期中的寄存器先写入后读取。假定在处理分支时, **预测它被选中**。如果所有存储器访问耗时 1 个周期, 这个循环的执行需要多少个周期?

Exercise C.1(D)

- ❖ Loop: LD R1,0(R2) ;load R1 from address 0+R2
- ❖ DADDI R1,R1,#1 ;R1=R1+1
- ❖ SD R1,0(R2) ;store R1 at address 0+R2
- ❖ DADDI R2,R2,#4 ;R2=R2+4
- ❖ DSUB R4,R3,R2 ;R4=R3-R2
- ❖ BNEZ R4,Loop ;branch to Loop if R4!=0
- ❖ 假定 R3 的初始值为 R2+396 。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LD R1,0(R2)	F	D	X	M	W													
DADDI R1,R1,#1		F	D	s	X	M	W											
SD R1,0(R2)			F	s	D	X	M	W										
DADDI R2,R2,#4					F	D	X	M	W									
DSUB R4,R3,R2						F	D	X	M	W								
BNEZ R4,Loop							F	s	D	X	M	W						
LD R1,0(R2)									F	D	X	M	W	W				

- ❖ Loop: LD R1,0(R2) ;load R1 from address 0+R2
- ❖ DADDI R1,R1,#1 ;R1=R1+1
- ❖ SD R1,0(R2) ;store R1 at address 0+R2
- ❖ DADDI R2,R2,#4 ;R2=R2+4
- ❖ DSUB R4,R3,R2 ;R4=R3-R2
- ❖ BNEZ R4,Loop ;branch to Loop if R4!=0
- ❖ 假定 R3 的初始值为 R2+396 。 预测它被选中

Exercise C.1(D)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LD R1,0(R2)	F	D	X	M	W													
DADDI R1,R1,#1		F	D	s	X	M	W											
SD R1,0(R2)			F	s	D	X	M	W										
DADDI R2,R2,#4					F	D	X	M	W									
DSUB R4,R3,R2						F	D	X	M	W								
BNEZ R4,Loop							F	s	D	X	M	W						
LD R1,0(R2)									F	D	X	M	W	W				

Again we have 99 iterations. There are two RAW stalls and a flush after the branch since the branch is taken. The total number of cycles is $8 * 98 + 12 = 796$.

高性能处理器拥有很深的流水线——超过 15 级。设想我们拥有一个 10 级流水线,其中 5 级流水线的每一级被分为 2 级。**唯一的难题是: 对于数据转发操作, 数据是由每一对流水级的末尾转发到需要这些数据的两个流水线的开头。例如, 数据从第二执行级的输出转发到第一执行级的输入, 仍然导致 1 个周期的延迟。**对于一个完整转发、旁路硬件的 10 级 RISC 流水线, 给出这一指令序列的时序。请使用如图 C-5 所示的流水线时序表。假定在处理分支时, 预测它被选中。如果所有存储器引用耗时 1 个周期, 这一循环的执行需要多少个周期?

- ❖ **Loop: LD R1,0(R2) ;load R1 from address 0+R2**
- ❖ **DADDI R1,R1,#1 ;R1=R1+1**
- ❖ **SD R1,0(R2) ;store R1 at address 0+R2**
- ❖ **DADDI R2,R2,#4 ;R2=R2+4**
- ❖ **DSUB R4,R3,R2 ;R4=R3-R2**
- ❖ **BNEZ R4,Loop ;branch to Loop if R4!=0**
- ❖ **假定 R3 的初始值为 R2+396 。 预测它被选中**

作者的参考答案没有考虑数据转发的延迟, BNEZ指令只考虑废弃一个周期
这里假设数据转发有延迟, BNEZ指令需要废弃3个周期。计算结果和作者的答案不同。

Exercise C.1(E)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
LD R1,0(R2)	F1	F2	D1	D2	X1	X2	M1	M2	W1	W2																	
DADDI R1,R1,#1		F1	F2	D1	D2	s	s	s	X1	X2	M1	M2	W1	W2													
SD R1,0(R2)			F1	F2	D1	s	s	s	D2	s	X1	X2	M1	M2	W1	W2											
DADDI R2,R2,#4				F1	F2	s	s	s	D1	s	D2	X1	X2	M1	M2	W1	W2										
DSUB R4,R3,R2					F1	s	s	s	F2	s	D1	D2	s	X1	X2	M1	M2	W1	W2								
BNEZ R4,Loop									F1	s	F2	D1	s	s	s	D2	X1	X2	M1	M2	W1	W2					
LD R1,0(R2)																	F1	F2	D1	D2	X1	X2	M1	M2	W1	W2	

Exercise C.1(E)

- ❖ Loop: LD R1,0(R2) ;load R1 from address 0+R2
- ❖ DADDI R1,R1,#1 ;R1=R1+1
- ❖ SD R1,0(R2) ;store R1 at address 0+R2
- ❖ DADDI R2,R2,#4 ;R2=R2+4
- ❖ DSUB R4,R3,R2 ;R4=R3-R2
- ❖ BNEZ R4,Loop ;branch to Loop if R4!=0
- ❖ 假定 R3 的初始值为 R2+396 。预测它被选中

作者的参考答案没有考虑数据转发的延迟，
BNEZ指令只考虑废弃一个周期
这里假设数据转发有延迟，BNEZ指令需要废弃
3个周期。计算结果和作者的答案不同。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
LD R1,0(R2)	F1	F2	D1	D2	X1	X2	M1	M2	W1	W2																	
DADDI R1,R1,#1		F1	F2	D1	D2	s	s	s	X1	X2	M1	M2	W1	W2													
SD R1,0(R2)			F1	F2	D1	s	s	s	D2	s	X1	X2	M1	M2	W1	W2											
DADDI R2,R2,#4				F1	F2	s	s	s	D1	s	D2	X1	X2	M1	M2	W1	W2										
DSUB R4,R3,R2					F1	s	s	s	F2	s	D1	D2	s	X1	X2	M1	M2	W1	W2								
BNEZ R4,Loop									F1	s	F2	D1	s	s	s	D2	X1	X2	M1	M2	W1	W2					
LD R1,0(R2)																	F1	F2	D1	D2	X1	X2	M1	M2	W1	W2	

We again have 99 iterations. There are three RAW stalls between the LD and ADDI, and one RAW stall between the DADDI and DSUB. Because of the branch prediction, 98 of those iterations overlap significantly. The total number of cycles is $16 * 98 + 22 = 1590$

- ❖ 假定在一个 5 级流水线中, 最长的流水级需要 0.8 ns, 流水线寄存器延迟为 0.1 ns 。这个 5 级流水线的时钟周期时间为多少? 如果 10 级流水线将所有流水级都分为两半, 那么 10 级机器的周期时间为多少呢?

Exercise C.1(F)

- ❖ 假定在一个 5 级流水线中, 最长的流水级需要 0.8 ns, 流水线寄存器延迟为 0.1 ns 。这个 5 级流水线的时钟周期时间为多少? 如果 10 级流水线将所有流水级都分为两半, 那么 10 级机器的周期时间为多少呢?

Exercise C.1(F)

- ❖ 0.9ns for the 5-stage pipeline and
- ❖ 0.5ns for the 10-stage pipeline.

- ❖ 利用第(d)、(e)部分的答案, 判断该循环在 5 级流水线和 10级流水线上的每指令周期数 (CPI)。利用第(f)部分计算的时钟周期, 计算每种机器的平均指令执行时间。

Exercise C.1(G)

- ❖ CPI of 5-stage pipeline: $796/(99*6) = 1.34$
- ❖ CPI of 10-stage pipeline: $1590/(99* 6) = 2.68$
- ❖ Avg Inst Exe Time 5-stage: $1.34 * 0.9 = 1.21 \text{ ns}$
- ❖ Avg Inst Exe Time 10-stage: $2.68 * 0.5 = 1.34 \text{ ns}$

- ❖ 利用第(d)、(e)部分的答案, 判断该循环在 5 级流水线和 10级流水线上的每指令周期数 (CPI)。利用第(f)部分计算的时钟周期, 计算每种机器的平均指令执行时间。

Exercise C.1(G)

Exercise C.2.1

- ❖ Suppose the branch frequencies (as percentages of all instructions) are as follows:
- ❖ Conditional branches 15%
- ❖ Jumps and calls 1%
- ❖ Taken conditional branches 60% are taken
- ❖ 假定分支频率如下所示(以占全部指令的百分比表示):
- ❖ 条件分支 15%
- ❖ 跳转与调用 1%
- ❖ 选中条件分支 60%被选中
- ❖ (a) We are examining a four-deep pipeline where the branch is resolved at the end of the second cycle for unconditional branches and at the end of the third cycle for conditional branches. Assuming that only the first pipe stage can always be done independent of whether the branch goes and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?(Keep two digits after the point)
- ❖ 我们正在研究一个深度为4的流水线，其中，无条件分支在第二周期结束时执行，而条件分支则在第三个周期结束时执行。假定仅第一个流水级总会完成，与是否执行该分支无关，忽略其他流水线停顿，在没有分支冒险的情况下，该机器的速度快多少？**预测它未被选中**



(a) We are examining a four-deep pipeline where the branch is resolved at the end of the second cycle for unconditional branches and at the end of the third cycle for conditional branches. Assuming that only the first pipe stage can always be done independent of whether the branch goes and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?(Keep two digits after the point)

Instruction	Clock cycle					
	1	2	3	4	5	6
i Jump	IF	ID	EX	WB		
i+1		stall	nop	nop	nop	...
Jump target			IF	ID	EXE	...

Instruction	Clock cycle					
	1	2	3	4	5	6
i Taken branch	IF	ID	EX	WB		
i+1		stall	IF	ID	nop	...
branch				IF	ID	...

Instruction	Clock cycle					
	1	2	3	4	5	6
i Not-taken branch	IF	ID	EX	WB		
i+1		stall	IF	ID	IF	

预测它未被选中

Exercise C.2.1

Exercise C.2.1

- ❖ Suppose the branch frequencies (as percentages of all instructions) are as follows:
 - ❖ Conditional branches 15%
 - ❖ Jumps and calls 1%
 - ❖ Taken conditional branches 60% are taken
 - ❖ (a) We are examining a four-deep pipeline where the branch is resolved at the end of the second cycle for unconditional branches and at the end of the third cycle for conditional branches. Assuming that only the first pipe stage can always be done independent of whether the branch goes and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?(Keep two digits after the point) 预测它未被选中
 - ❖ Pipeline speedup_{ideal} = 4
- | Control flow type | Frequency(per instruction) | Stalls(cycles) |
|-------------------------|----------------------------|----------------|
| Jumps and calls | 1% | 1 |
| Conditional (taken) | $15\% \times 60\% = 9\%$ | 2 |
| Conditional (not taken) | $15\% \times 40\% = 6\%$ | 1 |
- ❖ Pipeline stalls_{real} = $1 * 1\% + 2 * 9\% + 1 * 6\% = 0.25$
 - ❖ Pipeline speedup_{real} = $4 / (1 + 0.25) = 3.2$

Exercise C.2.2

- ❖ Suppose the branch frequencies (as percentages of all instructions) are as follows:
- ❖ Conditional branches 15%
- ❖ Jumps and calls 1%
- ❖ Taken conditional branches 60% are taken
- ❖ 假定分支频率如下所示(以占全部指令的百分比表示):
- ❖ 条件分支 15%
- ❖ 跳转与调用 1%
- ❖ 选中条件分支 60%被选中
- ❖ (b) Now assume a high-performance processor in which we have a 15-deep pipeline where the branch is resolved at the end of the fifth cycle for unconditional branches and at the end of the tenth cycle for conditional branches. Assuming that only the first pipe stage can always be done independent of whether the branch goes and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?(Keep two digits after the point)
- ❖ 现在假定有一个高性能处理器，其中有一个深度为15的流水线，无条件分支在第五周期结束时执行，条件分支在第十周期结束时执行。假定仅第一个流水级总会完成，与是否执行该分支无关，忽略其他流水线停顿，在没有分支冒险的情况下，该机器的速度快多少？**预测它未被选中**

Exercise C.2.2

- ❖ (b) Now assume a high-performance processor in which we have a 15-deep pipeline where the branch is resolved at the end of the fifth cycle for unconditional branches and at the end of the tenth cycle for conditional branches. Assuming that only the first pipe stage can always be done independent of whether the branch goes and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?(Keep two digits after the point)

Instruction	Clock cycle					
	1	2-5	6-10	11-15	16	17
i Jump	IF	ID	EX	WB		
i+1		stall	nop	nop	nop	...
Jump target			IF	ID	EXE	...

4 stalls

Instruction	Clock cycle						
	1	2-5	6-9	10	11-15	16	17
i Taken branch	IF	ID	EX		WB		
i+1		stall		IF	ID	nop	...
branch					IF	ID	...

9 stalls

Instruction	Clock cycle						
	1	2-5	6-9	10	11-15	16	17
i Not-taken branch	IF	ID	EX		WB		
i+1		stall		IF	ID	IF	

预测它未被选中
8 stalls

Exercise C.2.2

- ❖ (b) Now assume a high-performance processor in which we have a 15-deep pipeline where the branch is resolved at the end of the fifth cycle for unconditional branches and at the end of the tenth cycle for conditional branches. Assuming that only the first pipe stage can always be done independent of whether the branch goes and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?(Keep two digits after the point)
- ❖ 现在假定有一个高性能处理器，其中有一个深度为15的流水线，无条件分支在第五周期结束时执行，条件分支在第十周期结束时执行。假定仅第一个流水级总会完成，与是否执行该分支无关，忽略其他流水线停顿，在没有分支冒险的情况下，该机器的速度快多少？
- ❖ Pipeline speedup_{ideal} = 15
- ❖

Control flow type	Frequency(per instruction)	Stalls(cycles)
Jumps and calls	1%	4
Conditional (taken)	$15\% \times 60\% = 9\%$	9
Conditional (not taken)	$15\% \times 40\% = 6\%$	8
- ❖ Pipeline stalls_{real} = $4 * 1\% + 9 * 9\% + 8 * 6\% = 1.33$
- ❖ Pipeline speedup_{real} = $15 / (1 + 1.33) = 6.44$

控制冒险总结

- ❖ 控制冒险比数据冒险会引起更大的性能损失。
- ❖ 通常，流水线越深，在时钟周期上转移损失越大。
- ❖ CPI更高的处理器，会付出更高的转移代价。
- ❖ 预测机制的有效性取决于转移预测的准确性。