国防科技大学计算机学院

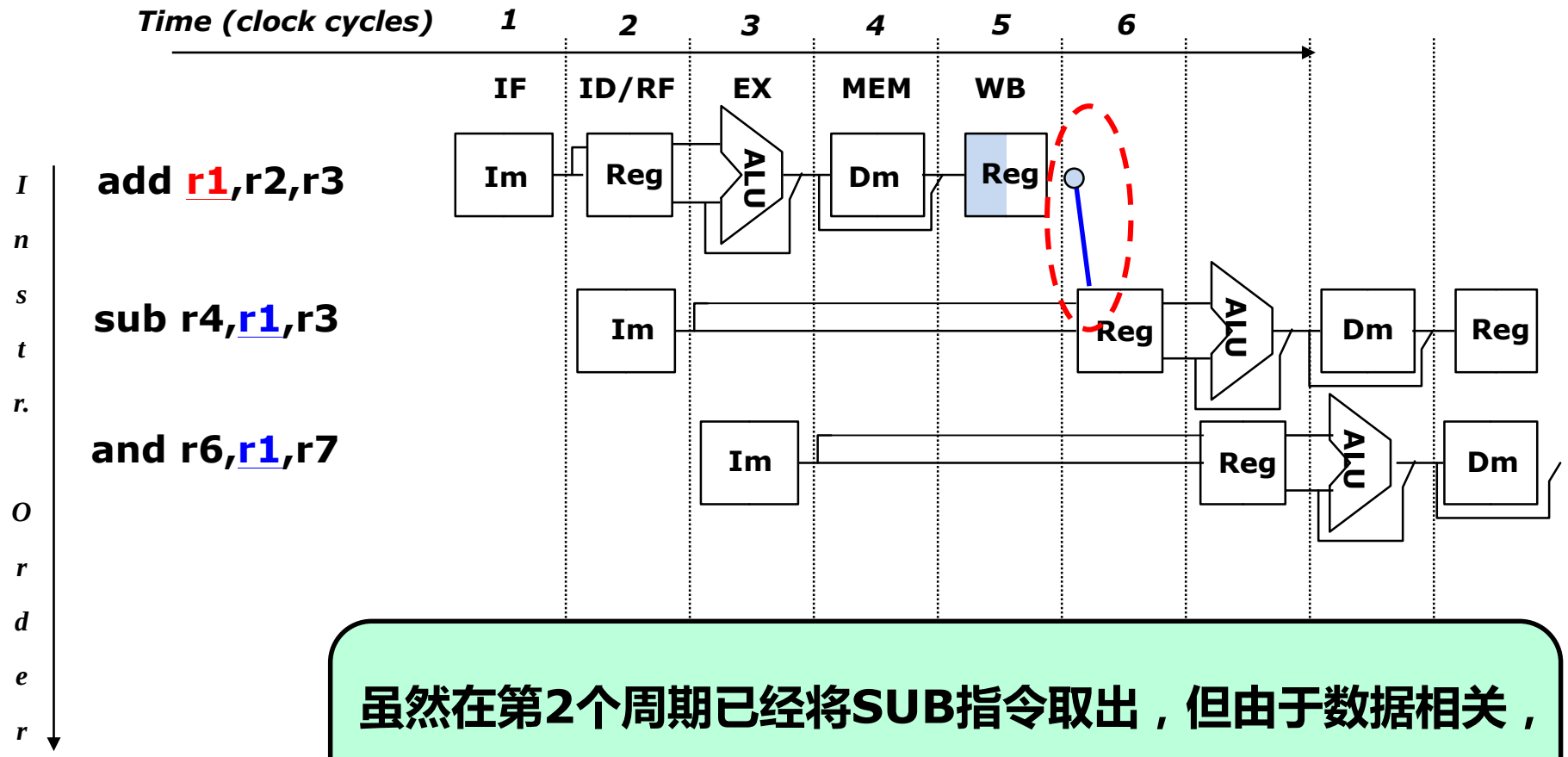
计算机原理

COMPUTER PRINCIPLE

第四章 第四节（3）数据冒险的解决方法

- 有四种方法可以解决流水线数据冒险
  - ① 硬件阻塞（stall）
  - ② 软件插入“NOP”指令
  - ③ 转发（Forwarding）或旁路（Bypassing）技术
  - ④ 编译优化：调整指令顺序

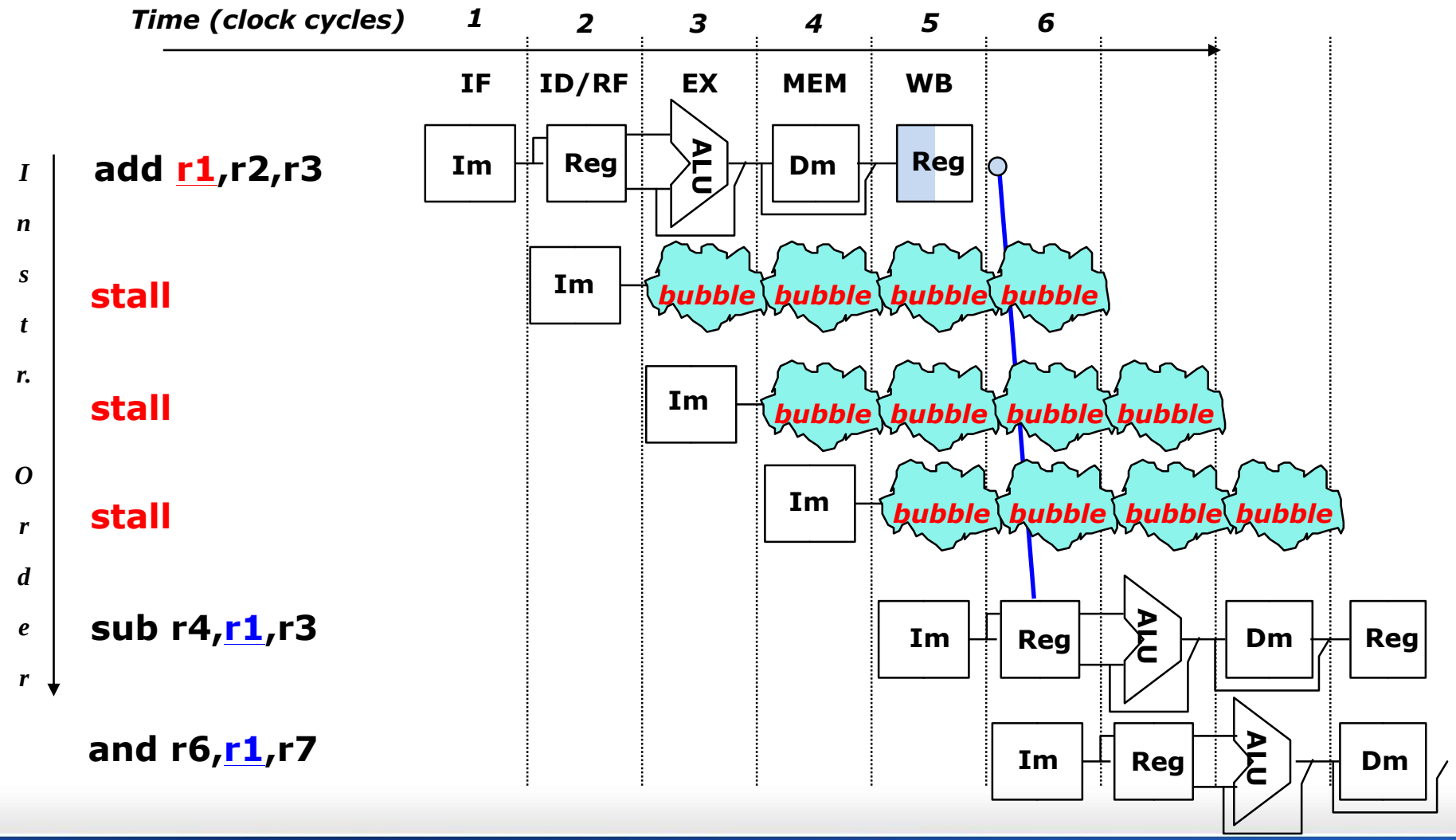
1. 方法一：硬件阻塞



虽然在第2个周期已经将SUB指令取出，但由于数据相关，SUB指令的译码最早只能在第6个周期进行。

1. 方法一：硬件阻塞

□ 硬件通过阻塞(stall)方式阻止后续指令执行，延迟到新值被写入寄存器以后！



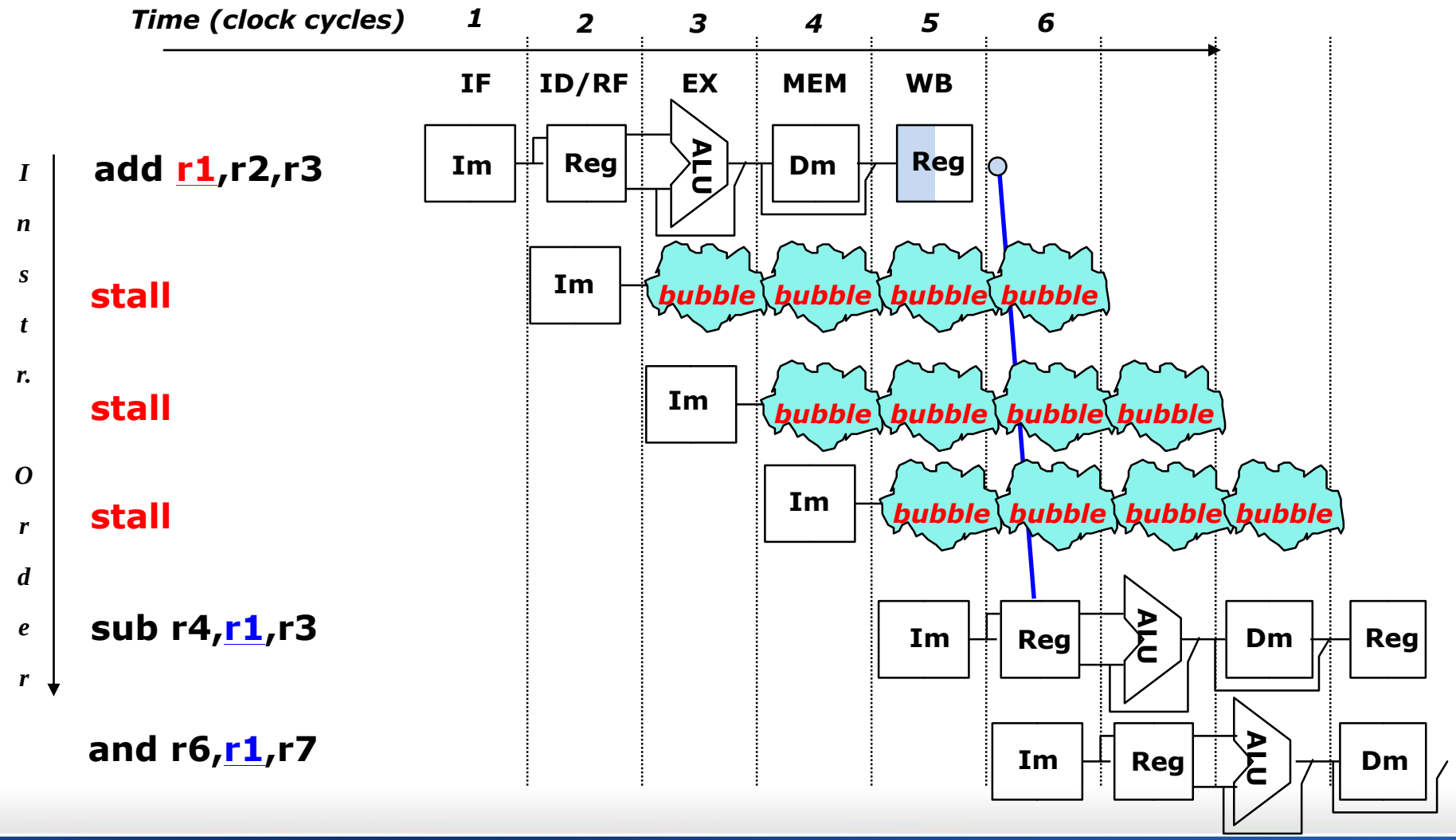
1. 方法一：硬件阻塞

□ 硬件通过阻塞(stall)方式阻止后续指令执行，延迟到新值被写入寄存器以后！

这种做法称为流水线阻塞，也称为“气泡bubble”。

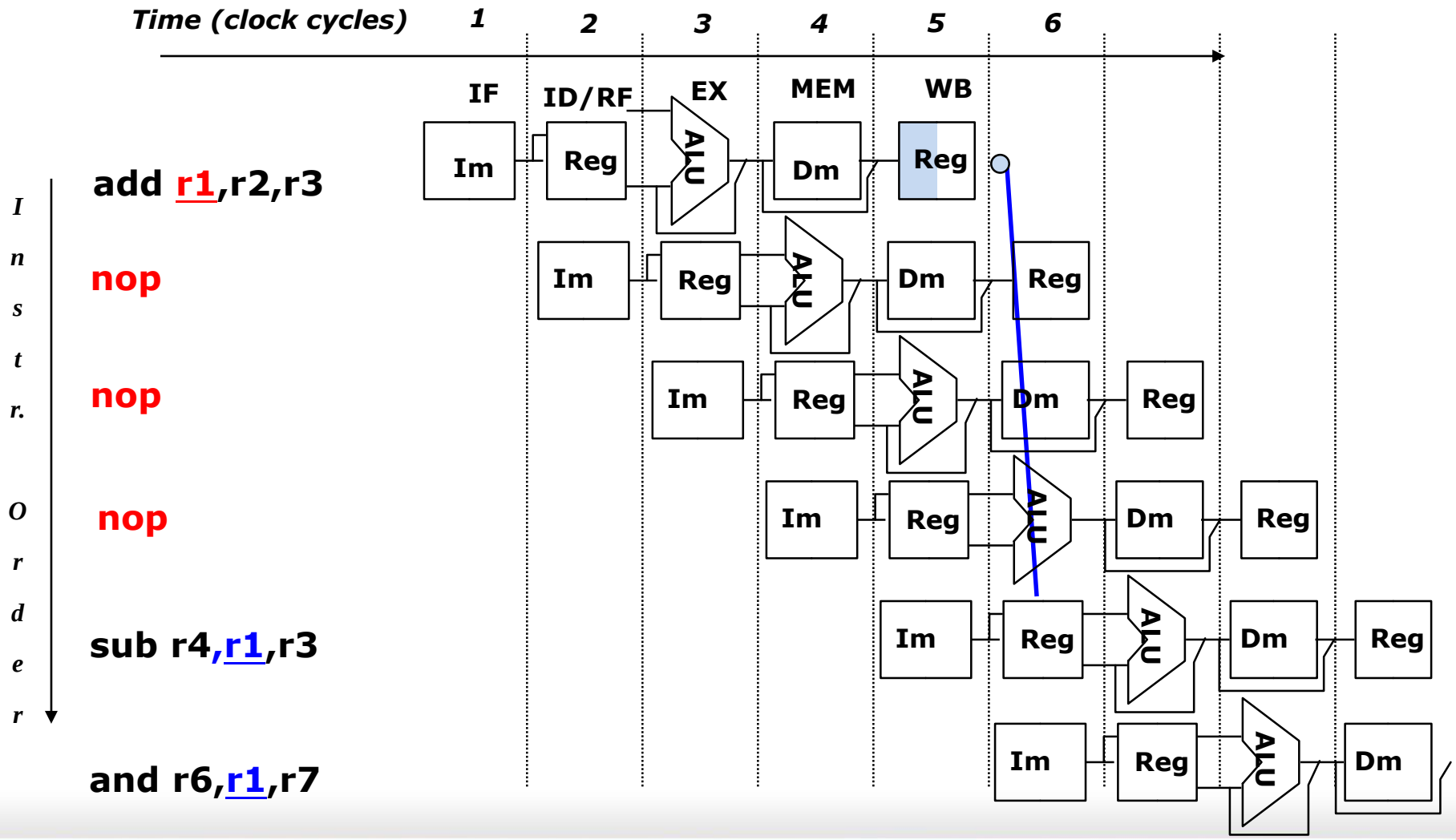
缺点：控制相当复杂，需要改数据通路！

- 检测数据相关
- 插入气泡



2. 方法二：软件插入“NOP”指令

❑ 在ADD与SUB指令之间增加若干条指令，也可以延迟SUB指令的执行。

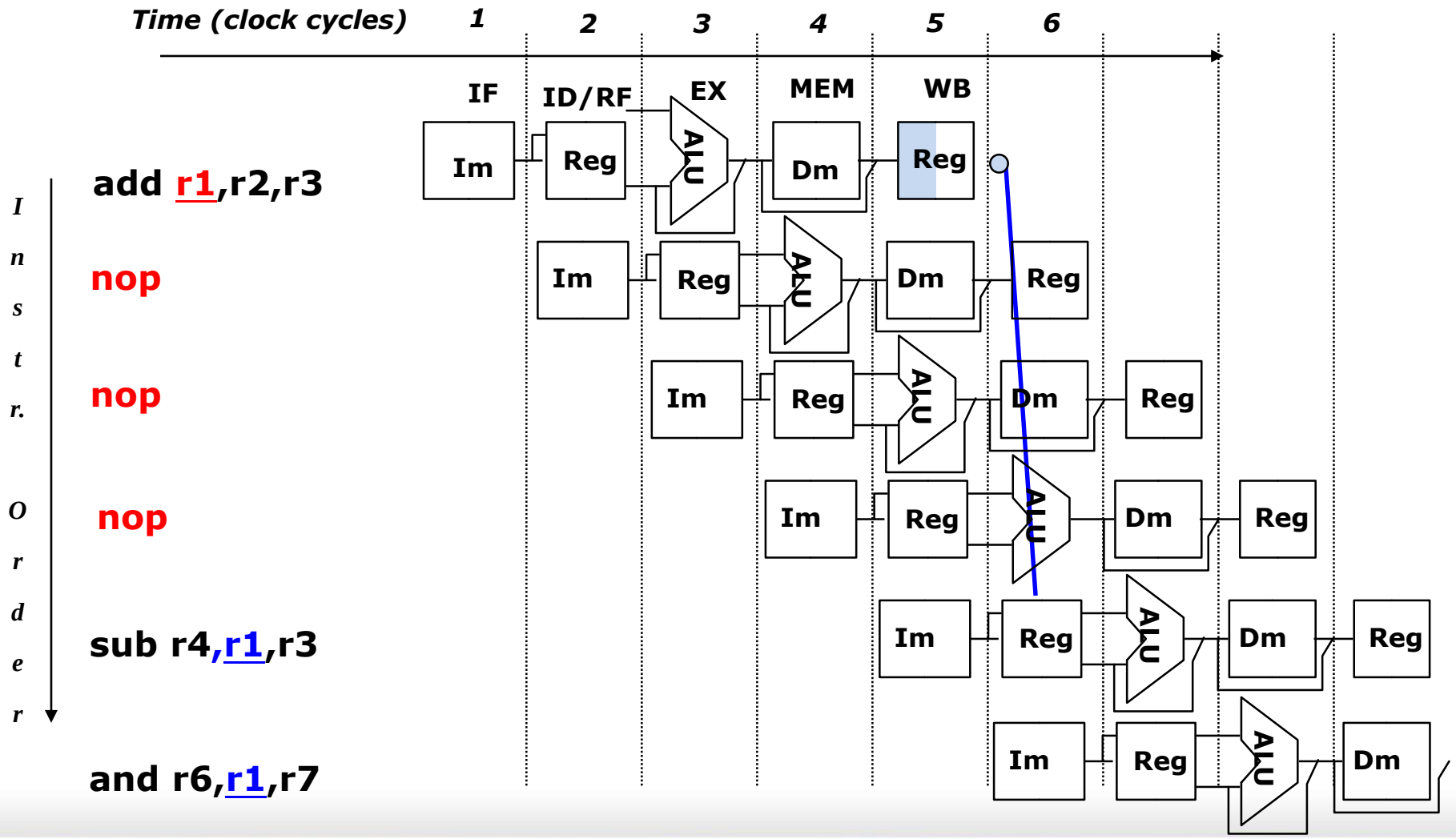


2. 方法二：软件插入“NOP”指令

❑ 在ADD与SUB指令之间增加若干条指令，也可以延迟SUB指令的执行。

这些指令不能影响程序的正确性，一般选择NOP指令。

缺点：浪费了三条指令的空间和时间！

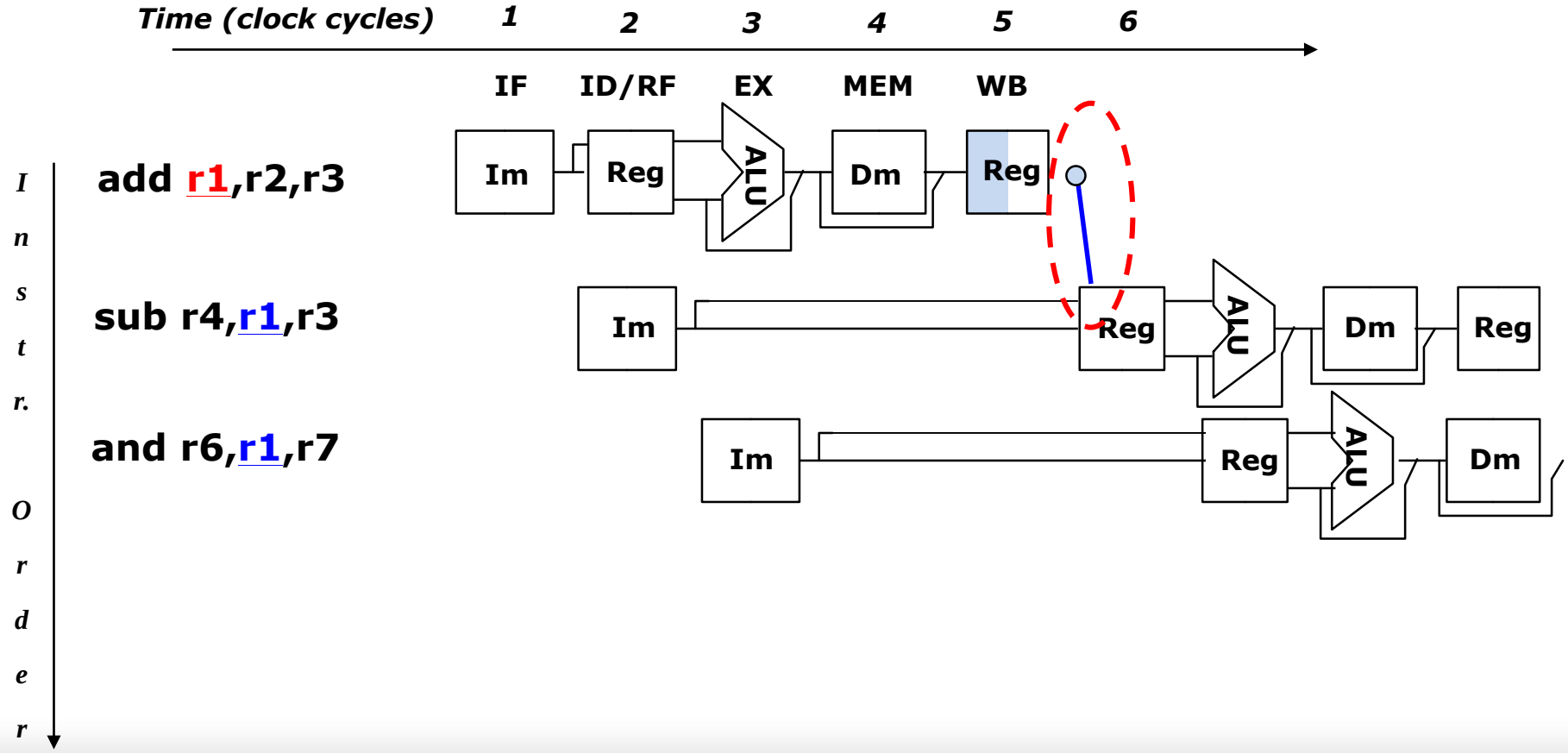


3. 方法三：转发（Forwarding）技术

- 指令SUB不能按时执行的原因
  - 寄存器中的数据不是最新的
  - 指令只能从寄存器中获得源操作数

在第4、5周期，最新数据已在流水线寄存器中

如果允许指令从流水线寄存器中获得源操作数呢？



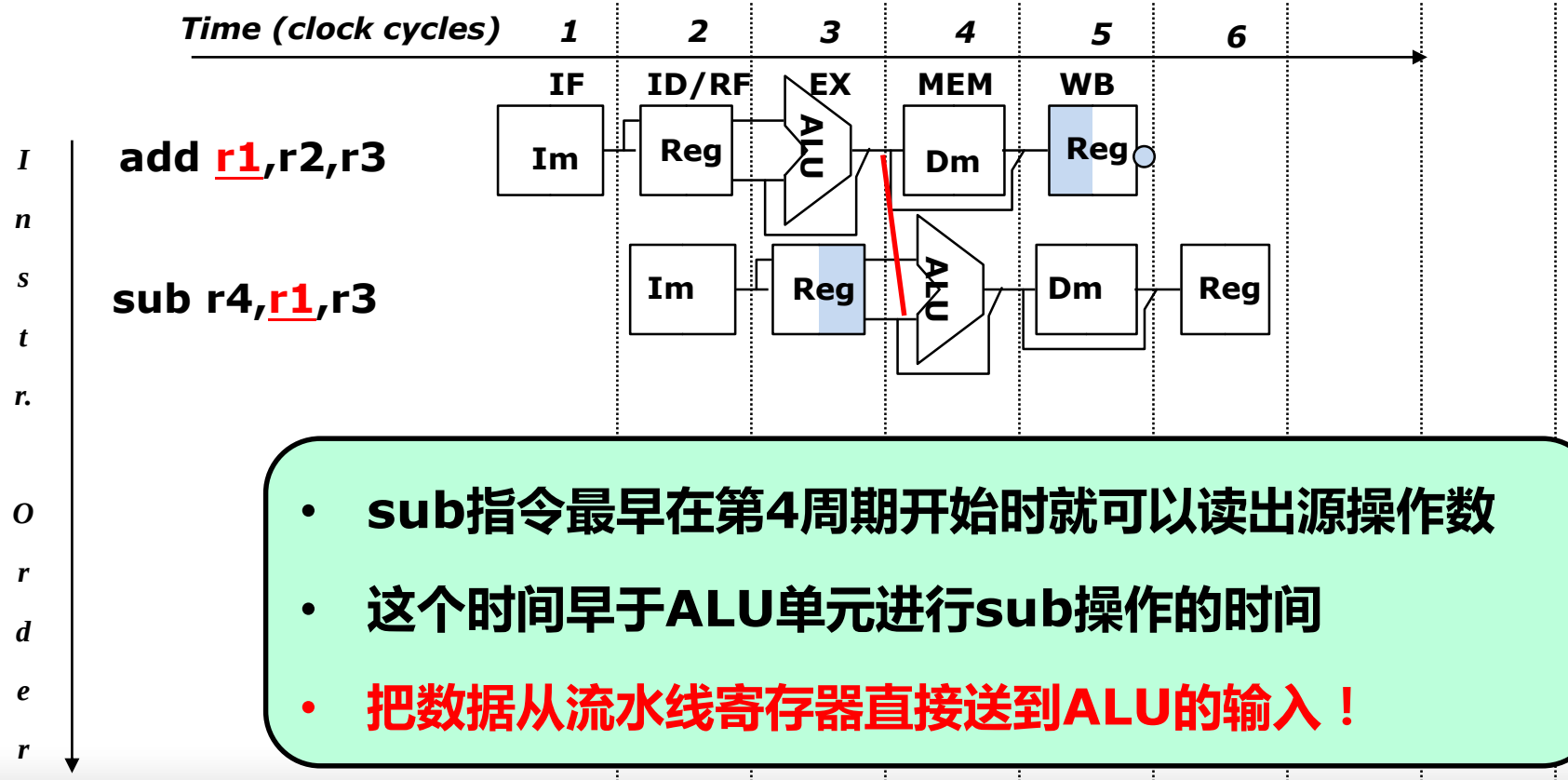
3. 方法三：转发（Forwarding）技术

❑ 指令SUB不能按时执行的原因

- 寄存器中的数据不是最新的
- 指令只能从寄存器中获得源操作数

在第4、5周期，最新数据已在流水线寄存器中

如果允许指令从流水线寄存器中获得源操作数呢？



- sub指令最早在第4周期开始时就可以读出源操作数
- 这个时间早于ALU单元进行sub操作的时间
- **把数据从流水线寄存器直接送到ALU的输入！**

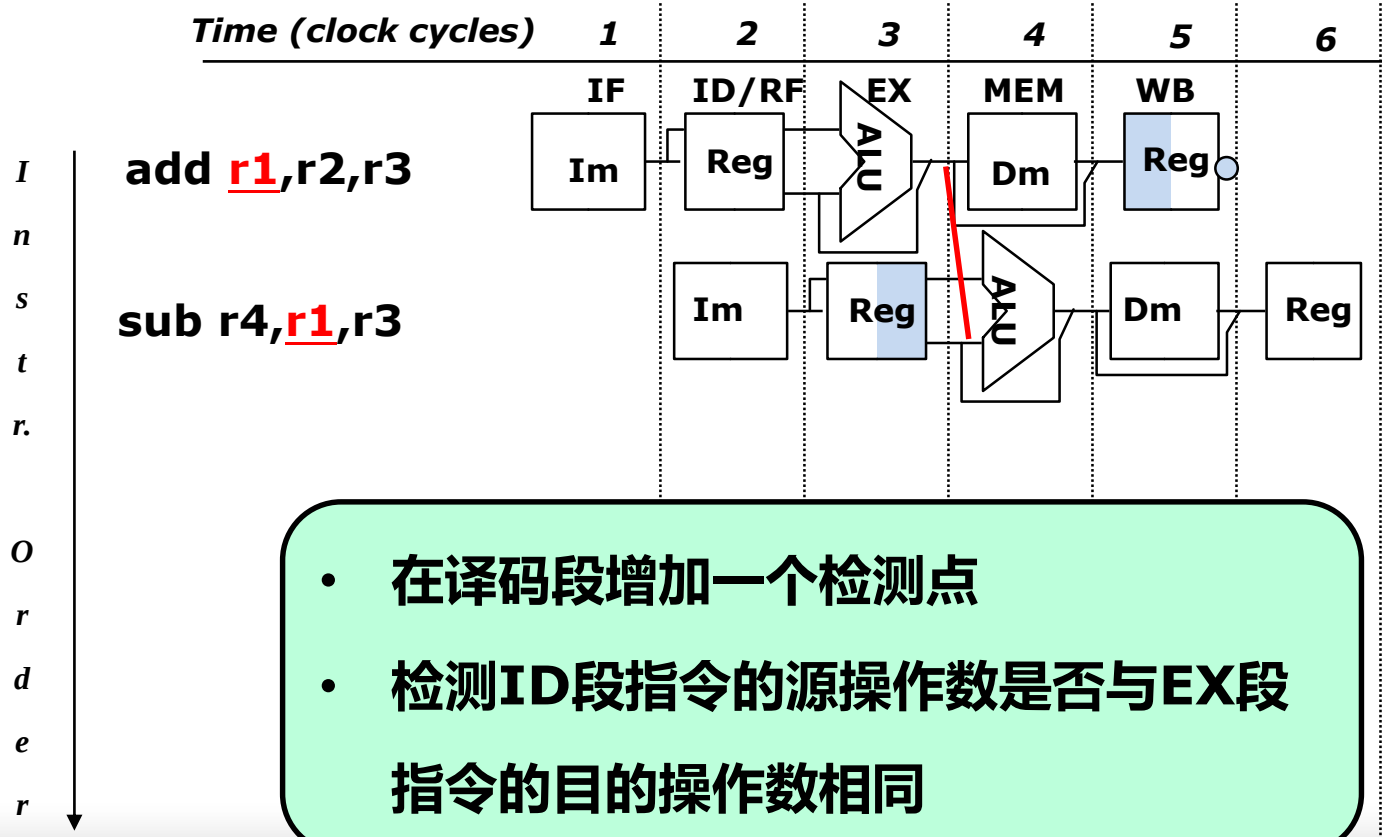
3. 方法三：转发（Forwarding）技术

❑ 指令SUB不能按时执行的原因

- ■ 寄存器中的数据不是最新的
- ■ 指令只能从寄存器中获得源操作数

在第4、5周期，最新数据已在流水线寄存器中

如果允许指令从流水线寄存器中获得源操作数呢？



- 在译码段增加一个检测点
- 检测ID段指令的源操作数是否与EX段指令的目的操作数相同

3. 方法三：转发（Forwarding）技术

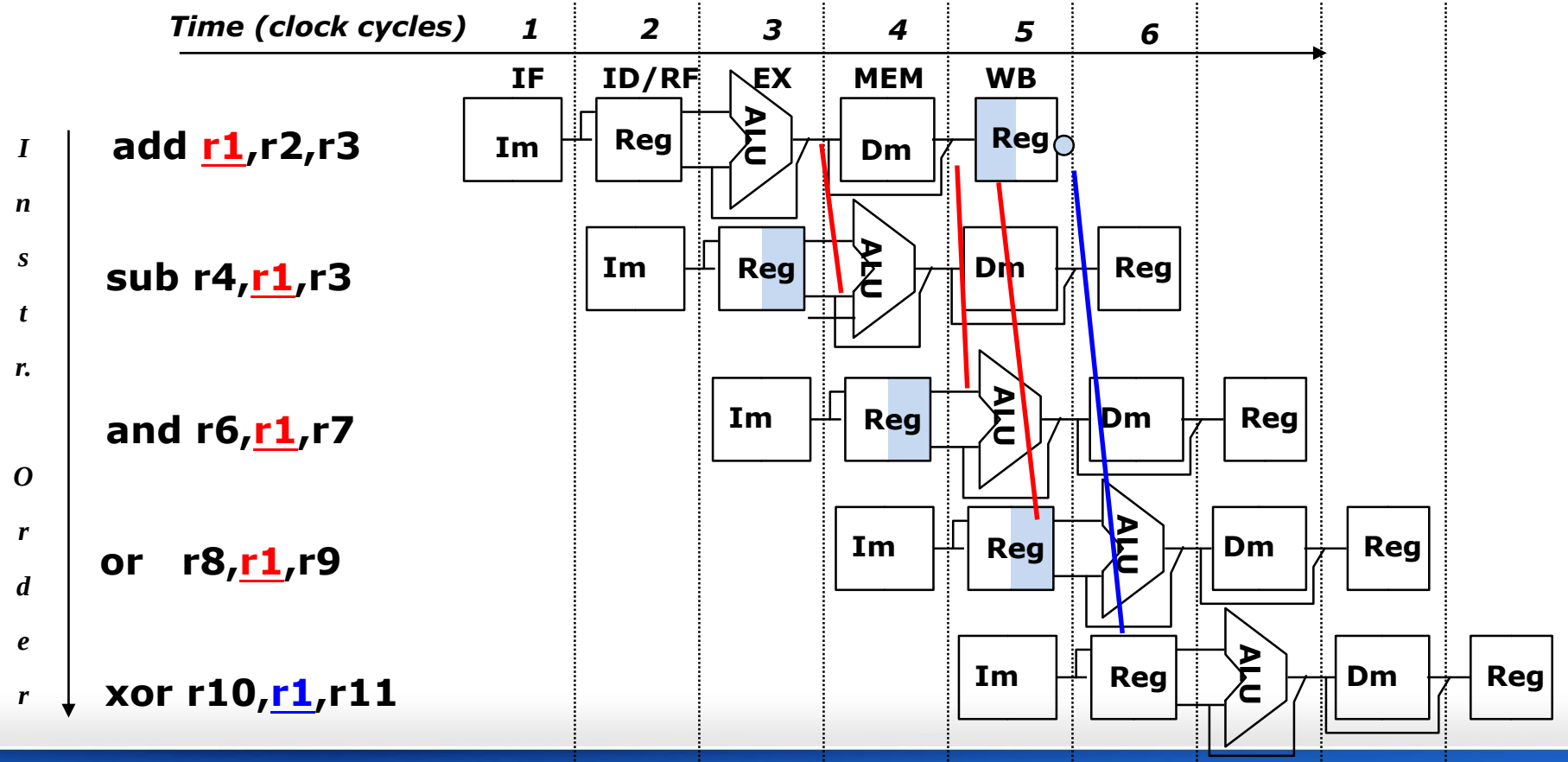
❑ 指令SUB不能按时执行的原因

- ■ 寄存器中的数据不是最新的
- ■ 指令只能从寄存器中获得源操作数

在第4、5周期，最新数据已在流水线寄存器中

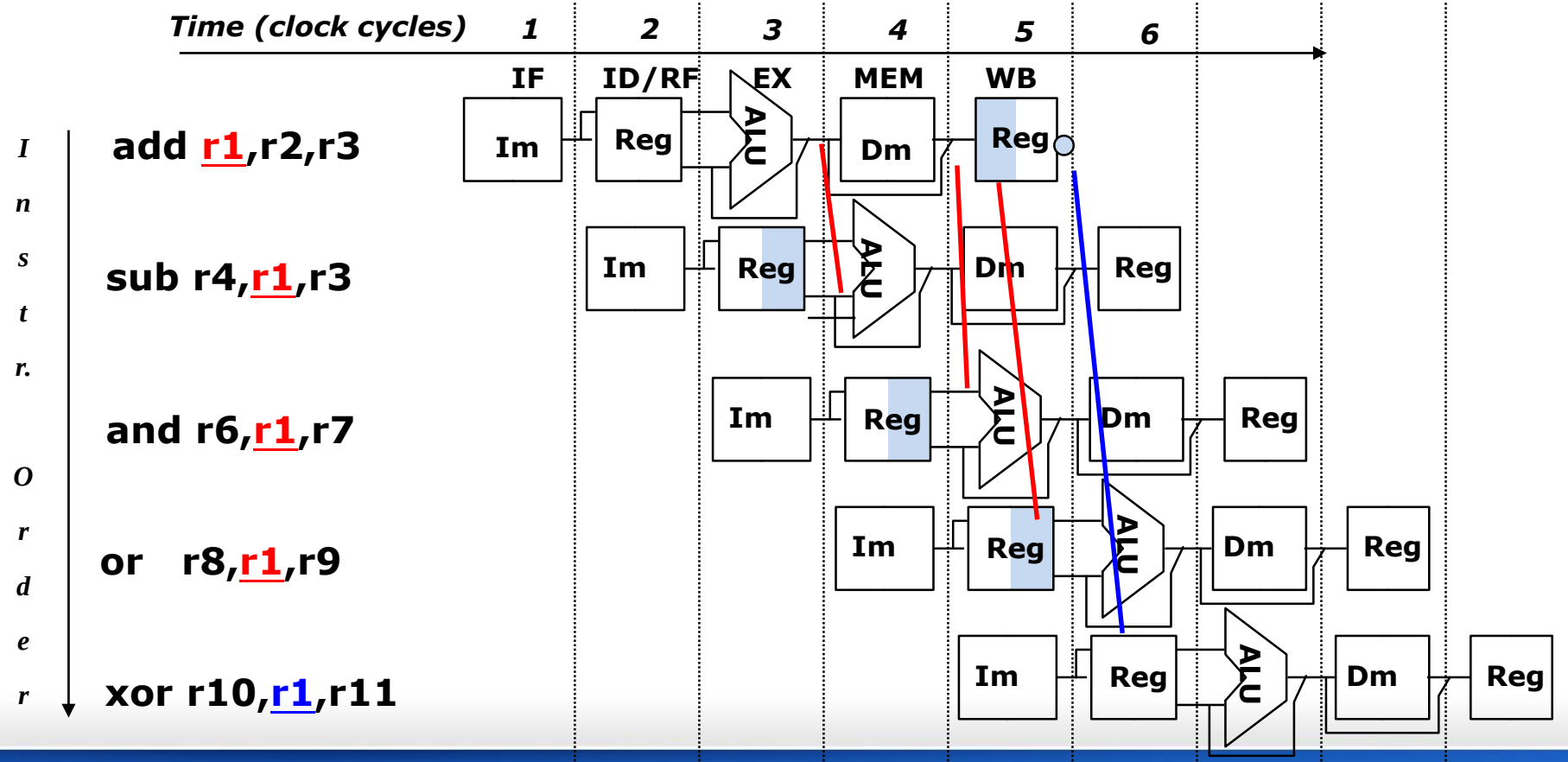
如果允许指令从流水线寄存器中获得源操作数呢？

- and和or指令也能利用这个机制。
- xor指令可以正常执行。



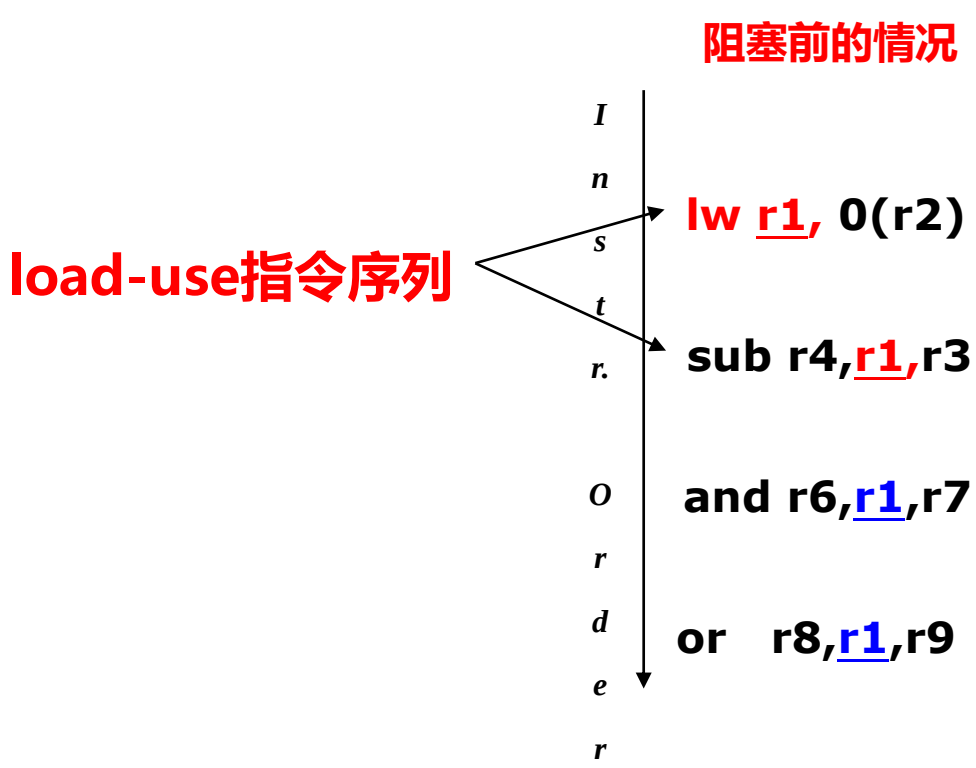
3. 方法三：转发（Forwarding）技术

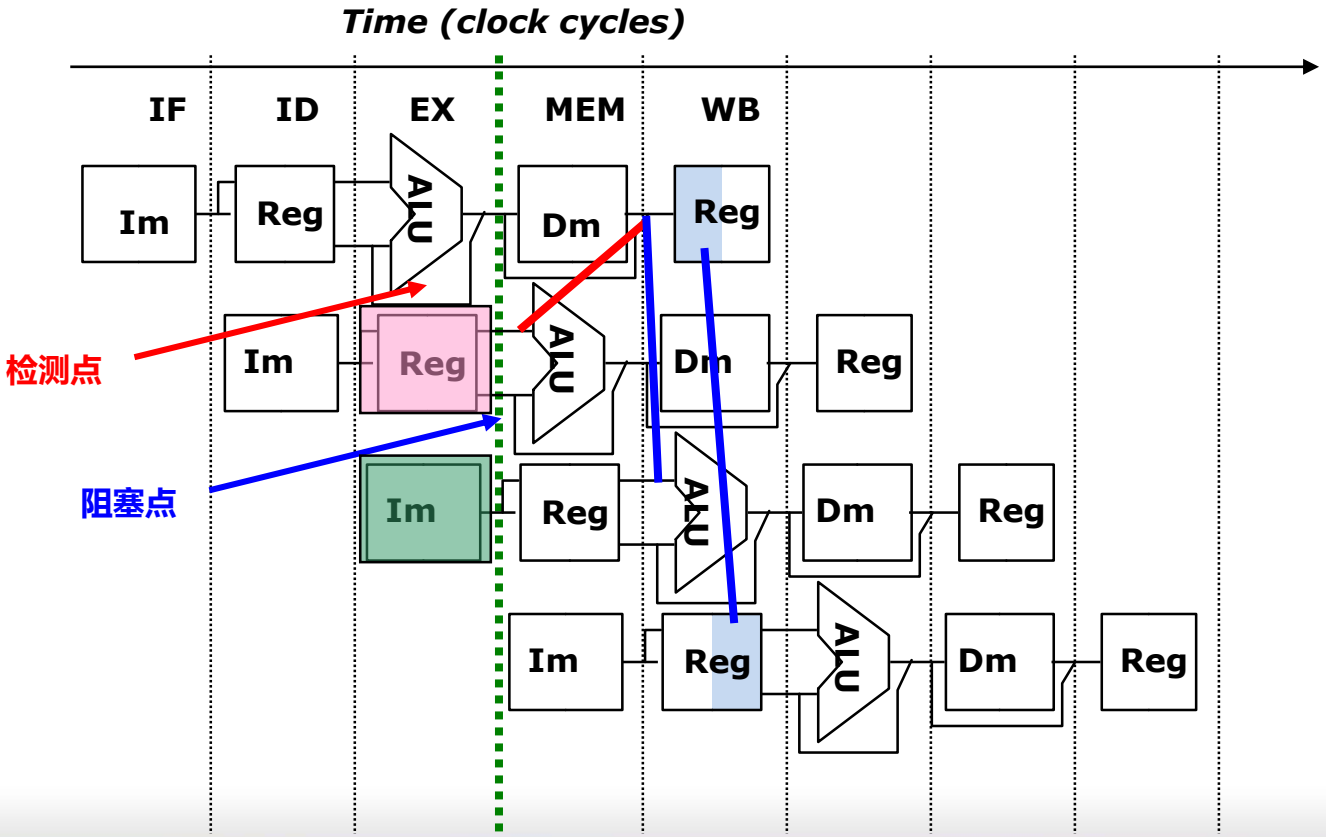
- 这种机制叫做转发(Forwarding)或旁路(Bypassing)
  - ① 把数据从流水线寄存器直接送到ALU的输入端
  - ② 寄存器的写/读操作分别在前/后半周期，写入的数据可在同一周期读出



3. 方法三：转发（Forwarding）技术

- 转发技术不是万能的！
  - 检测点：检测load-use指令序列(判断第一条指令是否为load)
  - 阻塞点：延迟use指令的执行



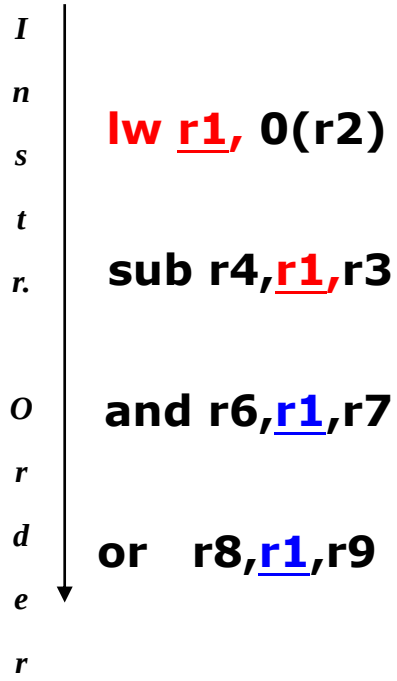


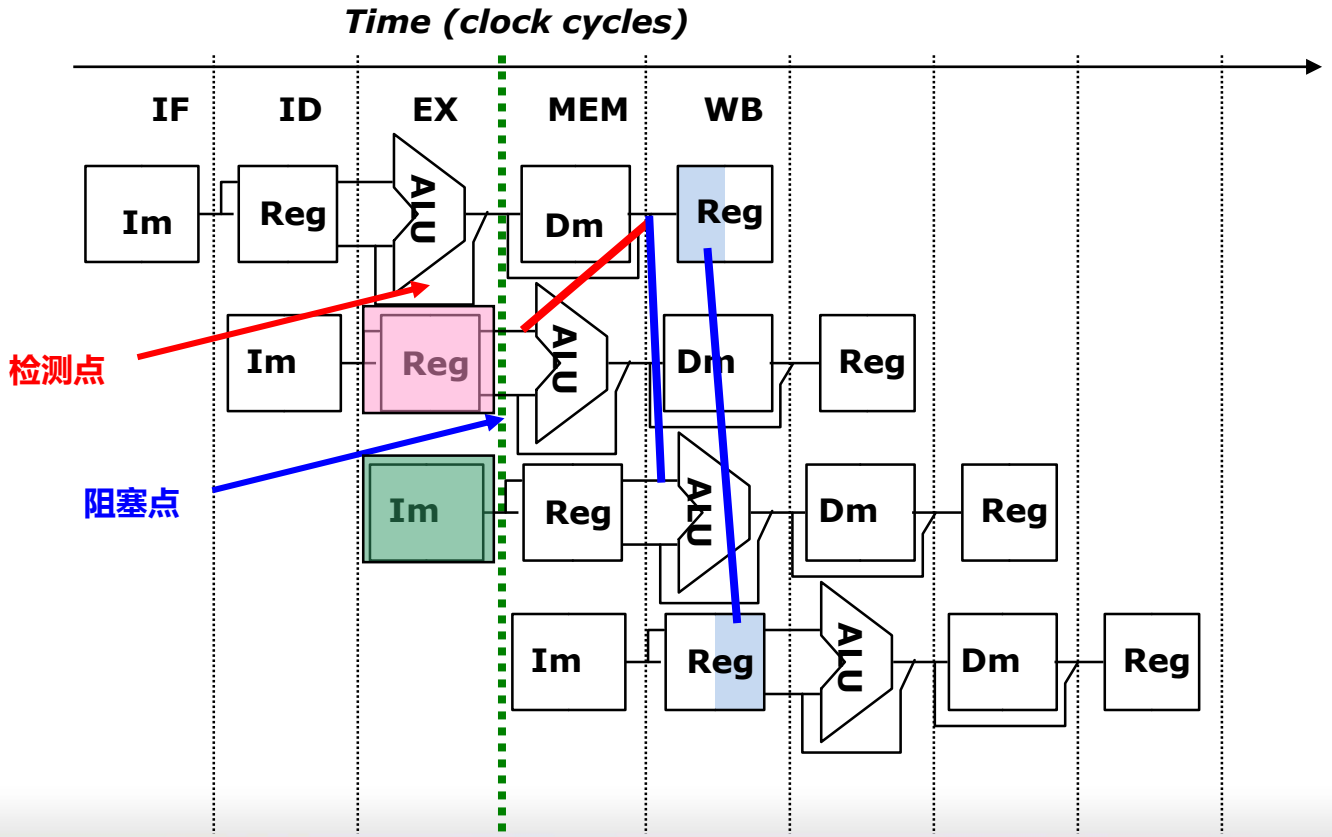
3. 方法三：转发（Forwarding）技术

- 转发技术不是万能的！
  - 检测点：检测load-use指令序列(判断第一条指令是否为load)
  - 阻塞点：延迟use指令的执行

- 在阻塞点，必须将sub和and指令的执行结果清除，并延迟一个周期执行这两条指令
- lw指令正常执行

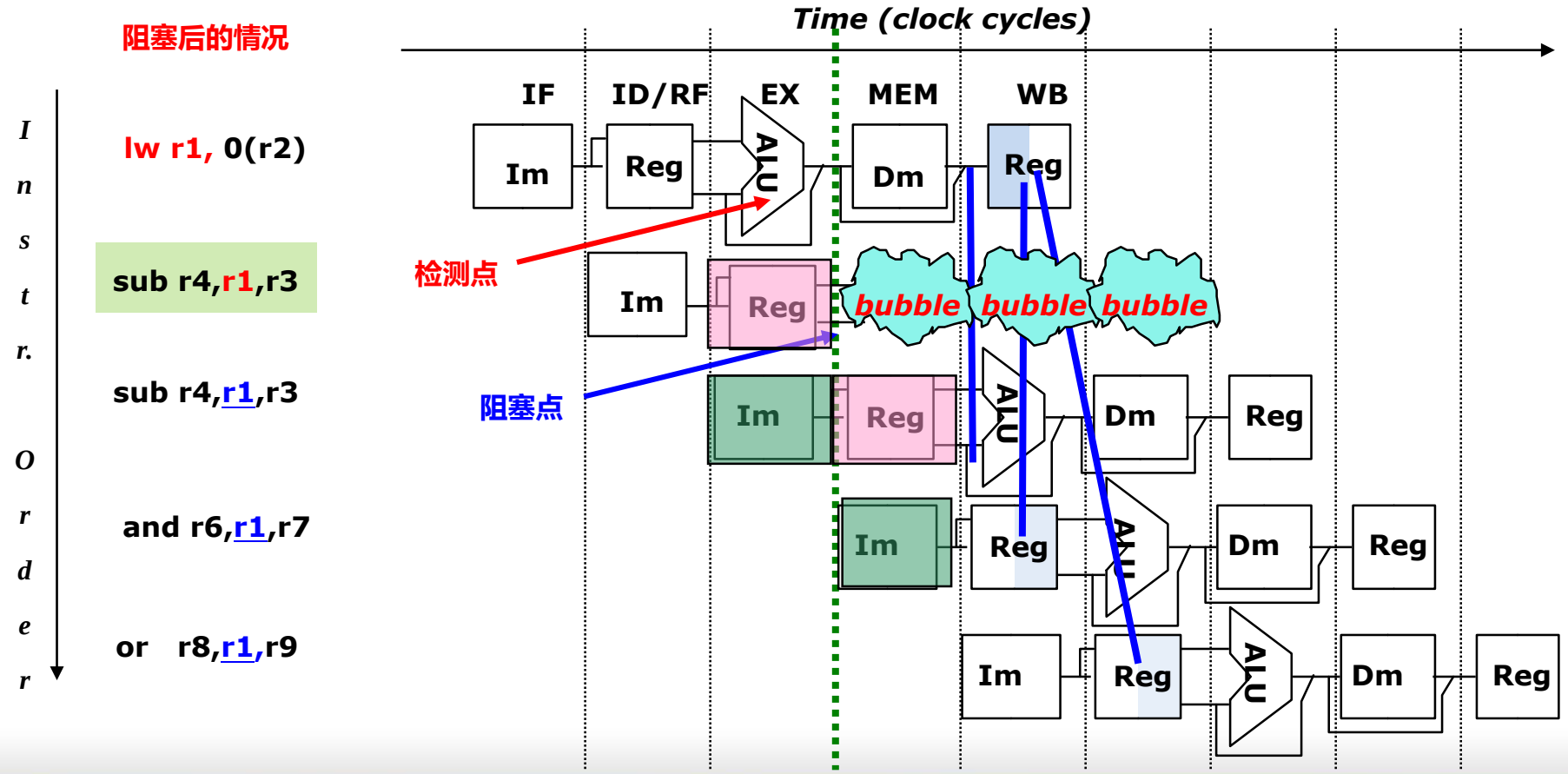
阻塞前的情况



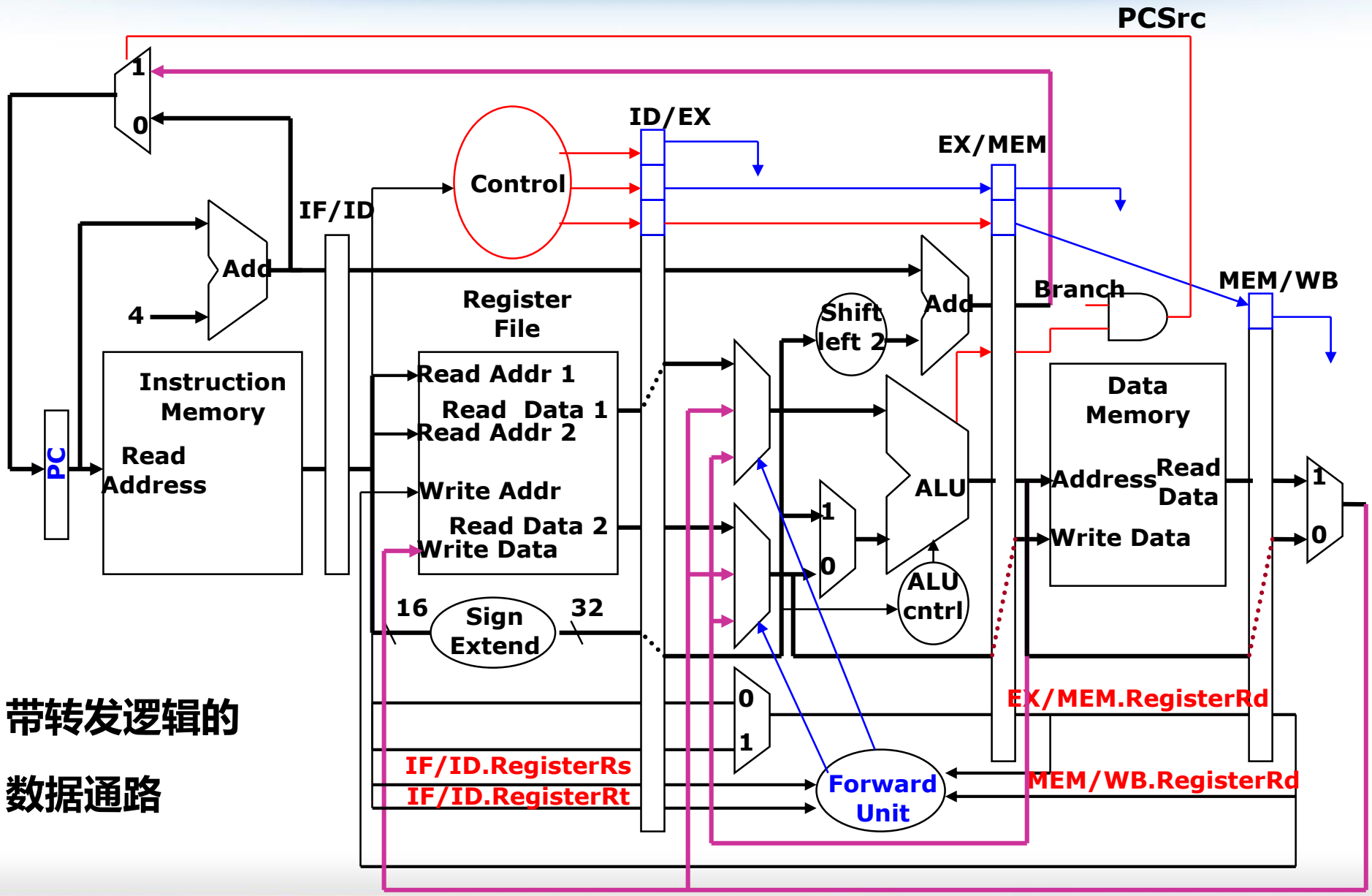


3. 方法三：转发（Forwarding）技术

- 转发技术不是万能的！
  - 检测点：检测load-use指令序列(判断第一条指令是否为load)
  - 阻塞点：延迟use指令的执行



3. 方法三：转发（Forwarding）技术



图：带转发逻辑的数据通路

4. 方法四：编译优化

例：以下源程序可生成两种不同的代码，优化代码可避免Load阻塞

a = b + c;

d = e – f;

假定 a, b, c, d ,e, f 在内存。

Slow code:

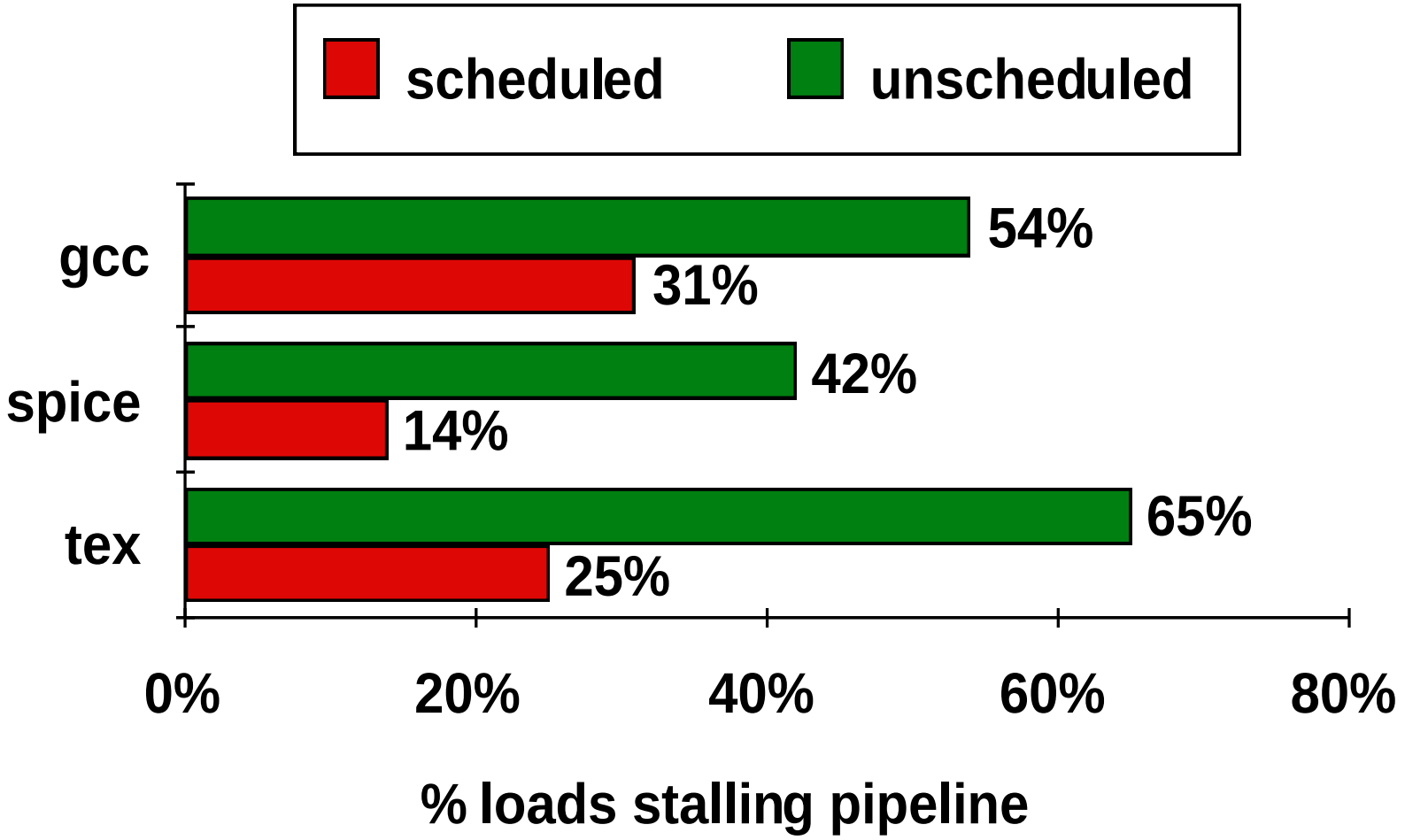
```
lw    $2, b
lw    $3, c
add   $1, $2, $3
sw    a, $1
lw    $5, e
lw    $6, f
sub   $4, $5, $6
sw    d, $4
```

调整后

Fast code:

```
lw    $2, b
lw    $3, c
lw    $5, e
add   $1, $2, $3
lw    $6, f
sw    a, $1
sub   $4, $5, $6
sw    d, $4
```

4. 方法四：编译优化



由此可见，优化调度后load-use阻塞现象大约降低了1/2~1/3