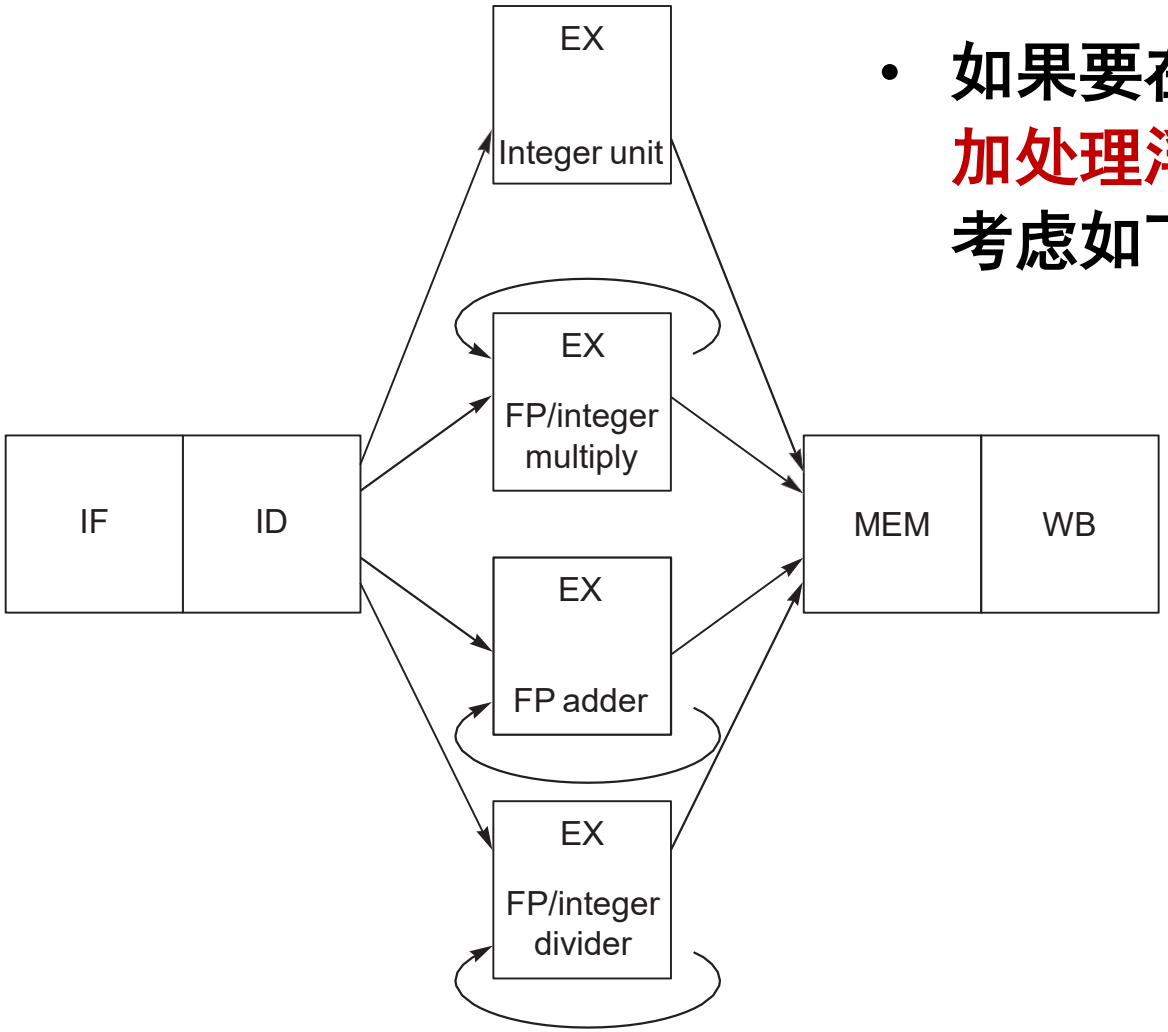
第3章 流水线模型机

- 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.5 扩展流水线到多执行周期操作

❖ 处理浮点操作的解决方案

- 在1或2个时钟周期完成浮点操作
  - 这意味着使用一个慢的时钟
  - 可能在一个FP部件中使用大量的逻辑电路
- 允许更长的操作时延
  - EX 级时钟周期可能需要重复很多次直到完成浮点操作
  - 可能有多个FP部件

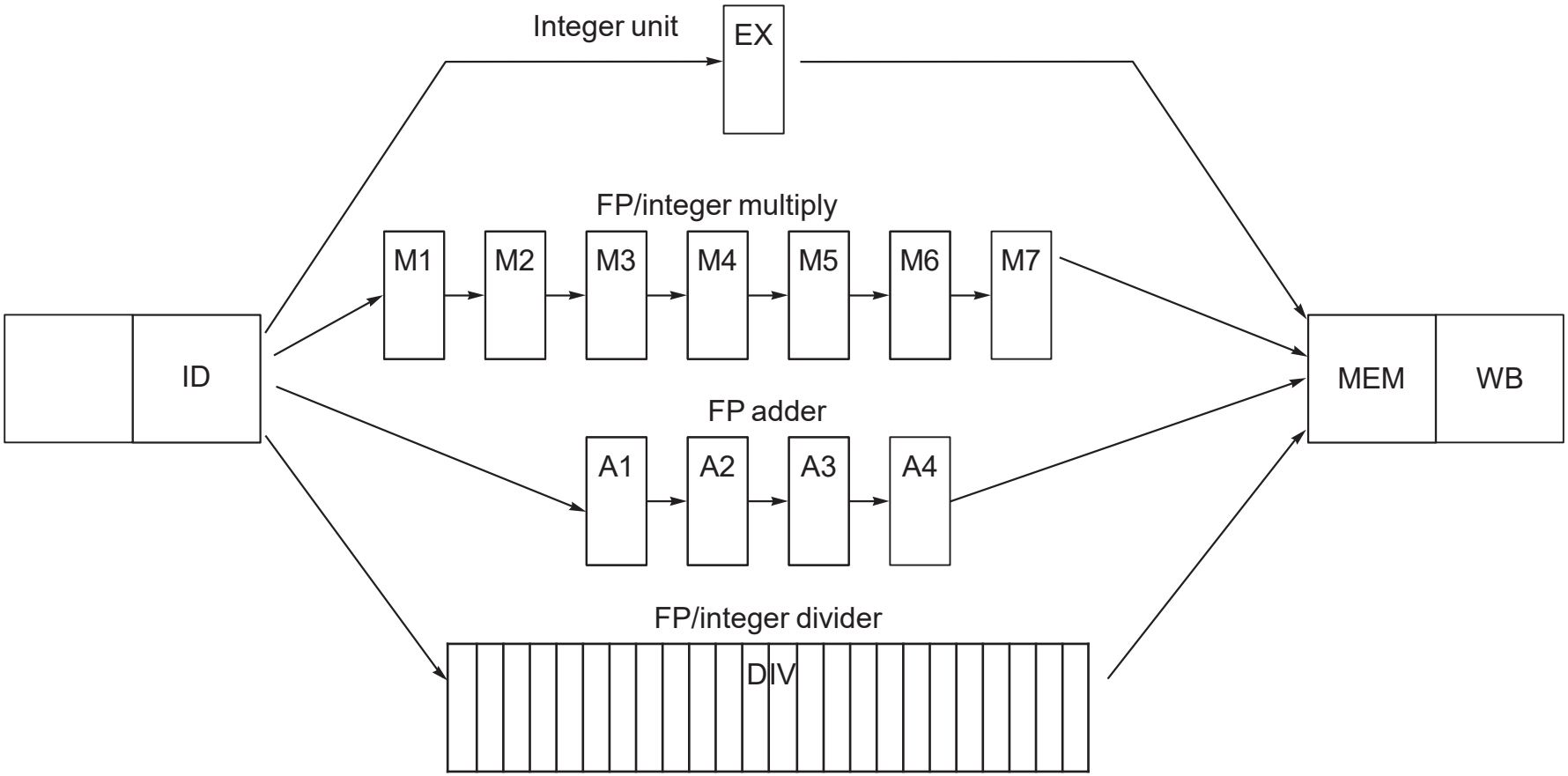


- 如果要在**整数流水线**中**增加处理浮点数**的功能，先考虑如下结构的流水线：

某些FP部件流水化

两个术语

- 延迟 Latency----在上一条指令生成结果之后，下一条指令能正常使用该结果而需等待的周期数（数据冒险停顿）
- 初始间隔Initiation interval----连续两条指令发射到同一个部件，指令操作之间必须间隔的时钟周期数（避免结构冒险）
  - 对于完全流水化的部件来说，初始间隔为 1
  - 对非流水线单元，初始间隔总是：延迟+1



Functional unit	Latency	Initiation interval
Integer ALU	0	1
FP multiply (also integer multiply)	6	1
FP add	3	1
FP divide (also integer divide)	24	25

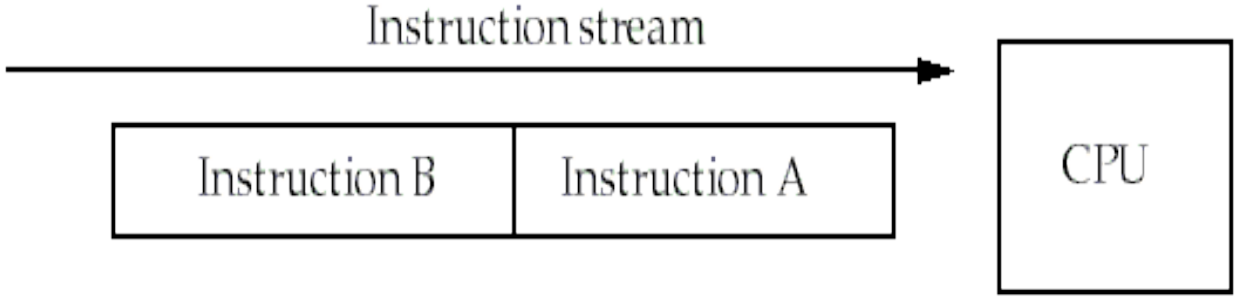
Figure C.34 Latencies and initiation intervals for functional units.

说明

- 需要新的流水线寄存器：
  - M1/M2，M2/M3，M3/M4，M4/M5，M5/M6，M6/M7
  - A1/A2，A2/A3，A3/A4
- 需要的连接寄存器：
  - ID/EX，ID/M1，ID/A1，ID/DIV
  - EX/MEM，M7/MEM，A4/MEM，DIV/MEM
- 除法部件是非流水线的：structural hazards
- 由于有不同执行时间的指令，同一时钟周期写寄存器的操作数目可能大于1
- 新的 data hazards：WAW，原因是乱序的 WBs
- 由于更长的操作延迟， RAW hazards 导致stalls更频繁

data hazards 的分类

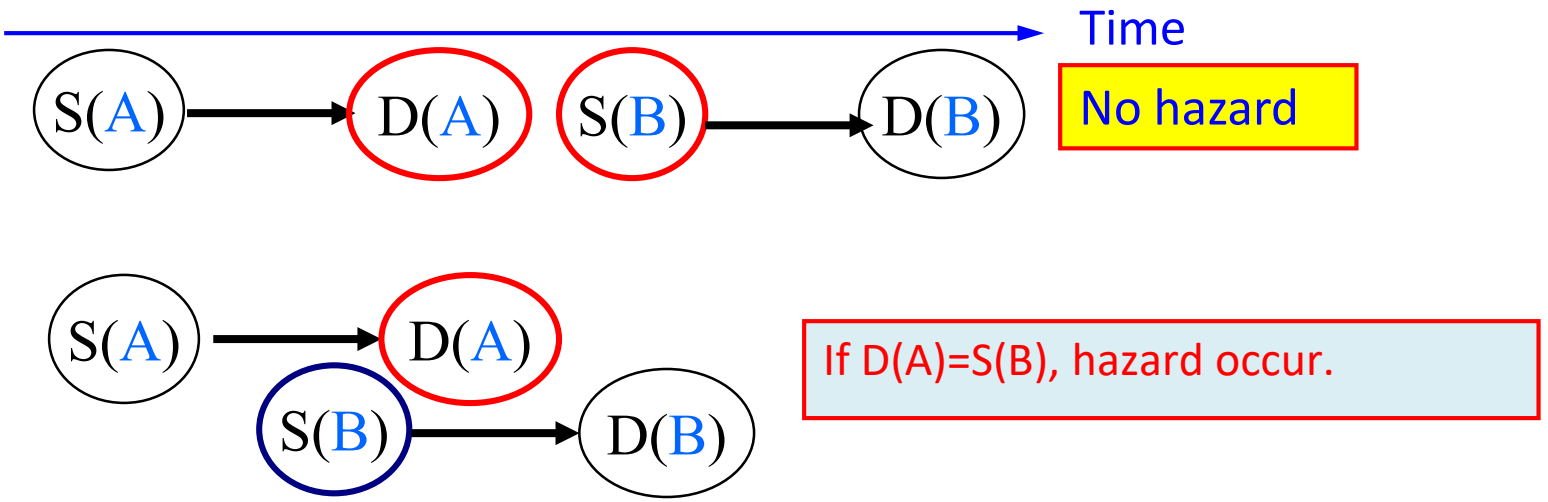
- 考虑两条指令，*A* 和 B。*A* 在B之前



- RAW(Read after write) 真相关 true dependence
  - 指令 A 写Rx，指令 B 读 Rx
- WAW(Write after write) 输出相关 output dependence
  - 指令 A 写Rx，指令 B 写 Rx
- WAR(Write after read) 反相关 anti-denpendence
  - 指令 A 读Rx，指令B 写 Rx
- 以上冒险的命名依据是数据操作必须在流水线上保持的顺序

RAW dependence

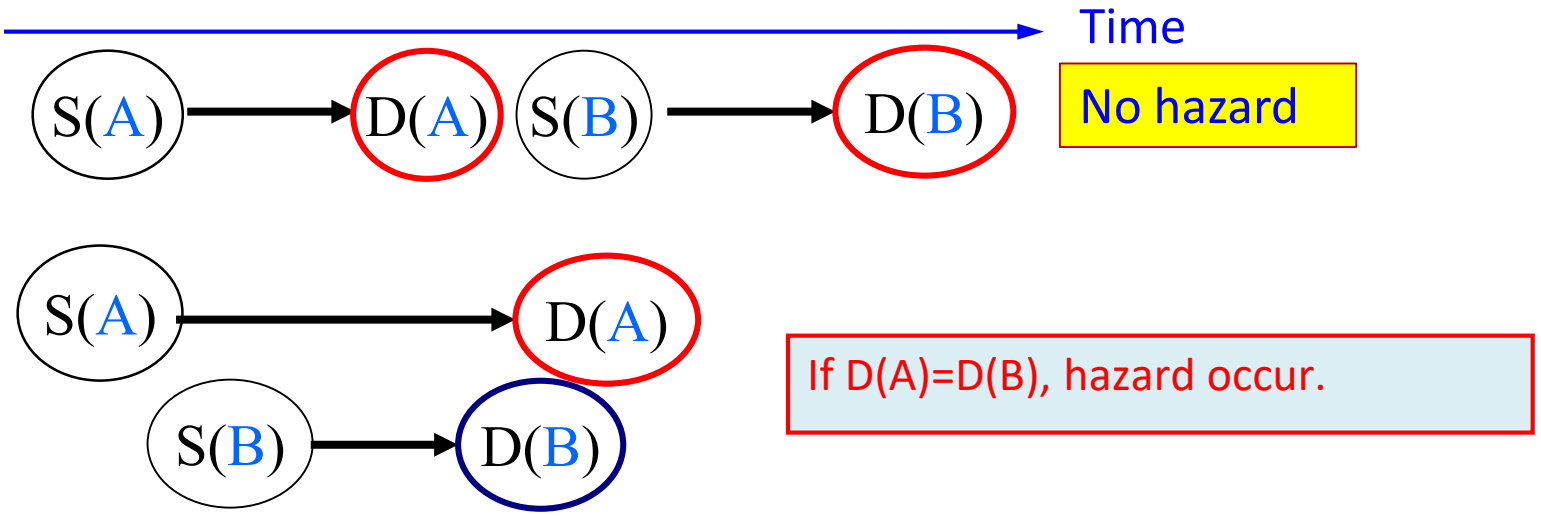
- B 试图在A写一个寄存器之前读它，得到旧的值。
- 经常发生，forwarding 可以帮助解决RAW



S表示源操作数；D表示目的寄存器

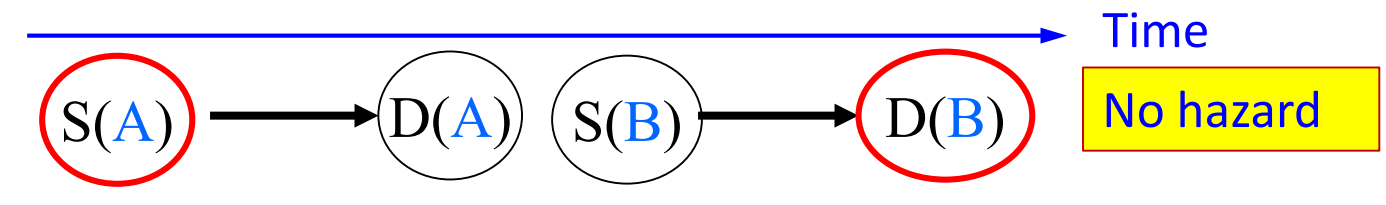
WAW dependence

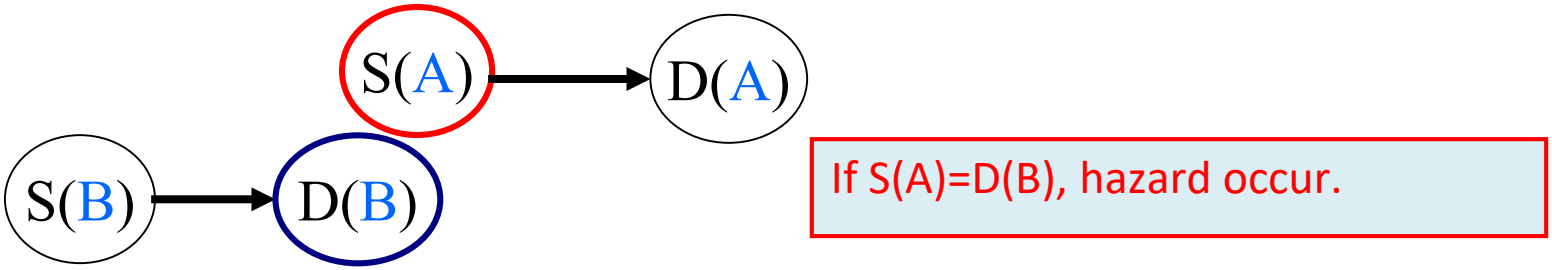
- B 试图在*A*写一个操作数之前写它。
- 在B执行后，寄存器应该是B的结果，但是*A*的结果取代了B的。
- 这种情况只发生在多于一个流水段写值的流水线，或长度可变的流水线如FP流水线。



WAR dependence

- B 试图在*A*读一个寄存器之前写它。
- 这种情况，*A*使用新（错误）的值。
- 很少发生，因为大多数流水线读值在前写值在后。
- 然而，可能发生在有复杂寻址方式的*CPU*中，如自增寻址。





维护精确异常处理

- 由于乱序完成，维护精确异常很困难
- Example

```
div.d    f0, f2, f4
add.d    f10, f10, f8
sub.d    f12, f12, f14
```

  - add.d 、sub.d 应该在div.d之前完成---乱序完成
  - 假定sub.d 产生一个运算异常时，add.d已经完成，但是div.d 还没有完成。
  - 结果是一个不精确的异常。这时的处理是让流水线排空（即等待div.d完成，再处理异常）。

更糟糕的情况

- 假定div.d 在add.d 完成后产生一个异常
  - 因为 add.d完成破坏了它的一个操作数，我们不能恢复到div.d指令之前的状态，即使用软件帮助也不行！

```
div.d     f0, f2, f4
add.d     f10, f10, f8
sub.d     f12, f12, f14
```

处理异常-- 第一种方案

- 忽略这个问题 (imprecise exceptions):
  - 快和容易，但是没有精确异常难以调试程序
  - 现代CPUs, i.e. DEC Alpha 21064、IBM Power-1 、MIPS R8000，提供了两种模式：
    * 一种是快速但可能不精确；
    * 另一种是慢速精确模式，某一时刻只允许一条浮点指令是活动的。

处理异常-- 第二种方案

❖ 缓存结果和延迟提交

- 保证在一条指令在完成之前，所有已经发射的指令操作不改变CPU的任何状态（写寄存器或存储器） 。
- 中间结果必须缓存（如果需要，还需缓冲forwarding的流水线寄存器）。
- 对于不同指令执行时间差异大的情况，需要缓存大量的中间结果，实现很困难。

处理异常—第三种方案

- 保持足够的信息，由处理程序为异常创建一个精确的序列：
  - 流水线上的指令和对应的*PCs*必须被保存
  - 在异常发生后，由软件完成执行，最后已执行完成指令之前未完成的所有指令
  - 这种方案用在SPARC architecture.

处理异常—第四种方案

- 在确认以前完成的指令没有引起异常后，才允许流水线上指令继续
  - 在EXE级的前期，浮点部件必须检测是否发生异常
  - 如果发生异常，暂停流水线以防止其后的指令完成，以维护精确中断
  - 这是 R4000 和 Pentium 的解决方案

为流水线设计指令系统的提示

- 尽量避免以下三种情况：
- 可变长指令和指令执行时间差异过大
  - 复杂化冒险检测和精确异常处理
- 复杂的寻址方式：如自增/减寻址会中间更新寄存器
  - 复杂化冒险检测和精确异常处理
- 隐含地设置条件码
  - 处理控制冒险时，显示设置条件码有利于指令重新调度
  - 复杂化冒险检测和精确异常处理

本章小结

- 流水线的相关概念
- 流水线定义、流水线冒险及处理
- 流水线模型机指令系统、无相关流水线模型机的设计与实现
- 流水线模型机中的相关处理
- 精确异常、非精确异常
- 浮点流水线的结构、相关问题及处理