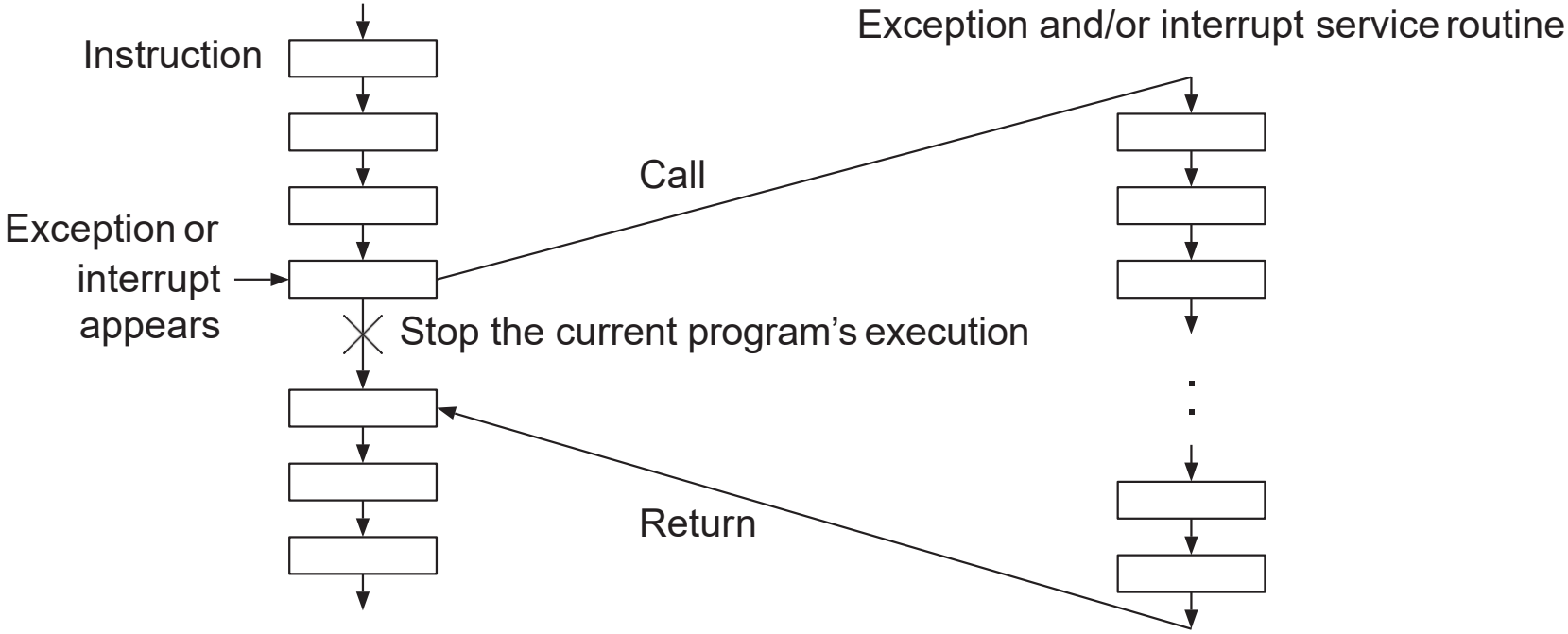
第3章 流水线模型机

- 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

异常和中断

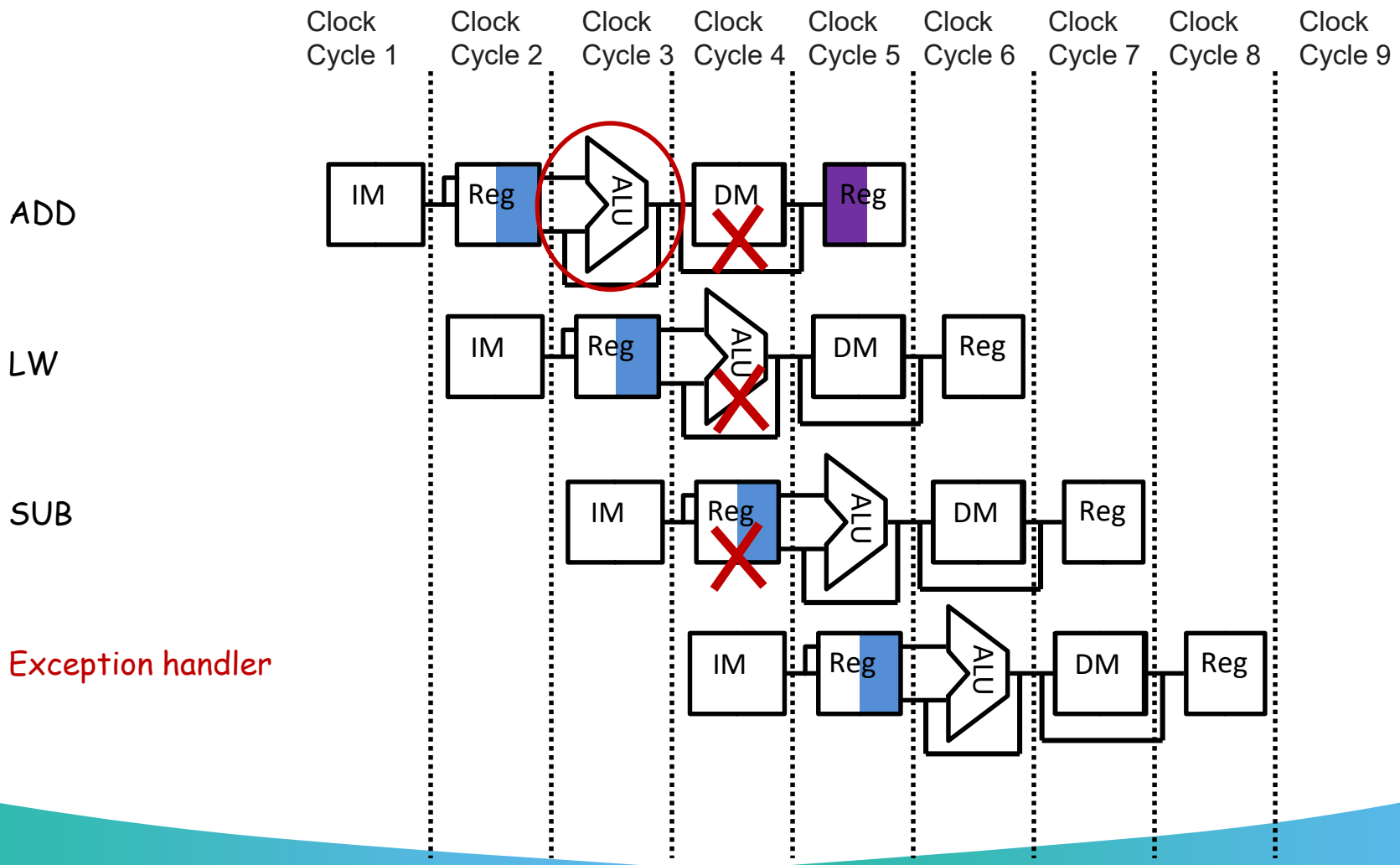
- 异常就是*中断*正常程序执行的*异常*事件
- 不同机器对异常的描述有差别，一般称为中断和异常。
- 中断例子： I/O 外设请求，如
  - 用户敲击键盘
  - 网络包到达
- 异常例子
  - 用户 OS 服务请求
  - 断点
  - 整数算术运算溢出
  - 除0
  - 浮点算术异常
  - 缺页
  - 未对齐的存储器访问
  - 违反了存储器保护权限
  - 硬件故障
  - 未定义指令

Dealing with exceptions and interrupts



在流水线上出现了异常的情况是怎样的?

- Example: Add instruction **overflows** in clock cycle 3



异常发生时该怎样做?

- 通常，异常发生在多条指令重叠执行时
  - 例如：load指令在MIPS的"MEM流水段" 可能发生缺页故障
  - 异常发生时，禁止异常指令及其后已经进入流水线指令的所有写操作，避免后续指令改变机器出错时的状态
  - 转异常处理，然后重新启动引起异常的指令或它的下一条指令

精确异常 (Precise Exceptions)

- 如果流水线停下来使异常指令之前的指令能正常结束，异常指令及其之后的指令能重新启动，则称该流水线是精确异常。
  - 异常指令之前的所有指令正常完成
  - 异常指令及其后进入流水线的指令没有改变机器的状态。
- 在这个模型下，重新执行就很简单了：
  - 重新恢复执行异常指令
  - 如果它不是一个可恢复执行的指令，则执行下一条指令

非精确异常

- 当不同指令执行时钟周期数有多种时，难以实现精确异常。
  - 在某条指令产生异常之前，后面的指令可能已经执行完毕
  - 例如

```
mul r1, r2, r3 ; 需 10 cycles
add r10,r11,r12 ; 需 5 cycles
```

  - add指令在mul指令前执行完成。 如果mul溢出，
  - 一个异常将出现在add指令已经更新了r10值之后。
  - 这是由于指令的乱序完成造成的，add在mul完成之前执行完毕。
  - 异常发生后，要恢复mul执行前状态很困难，这种情况为非精确异常

精确 vs 非精确异常

- 某些流水线处理器实现了两种工作模式：精确异常和非精确异常
  - 特殊的软件指令保证精确异常
  - 处理器若工作在精确异常模式下，运行速度会更慢。
  - 一般，RISC整数操作容易实现精确异常，而浮点异常一般不容易实现精确异常的。

整数MIPS架构中的异常

- 异常可能在流水线的哪些阶段中产生？
- 阶段　　可能产生的异常类型

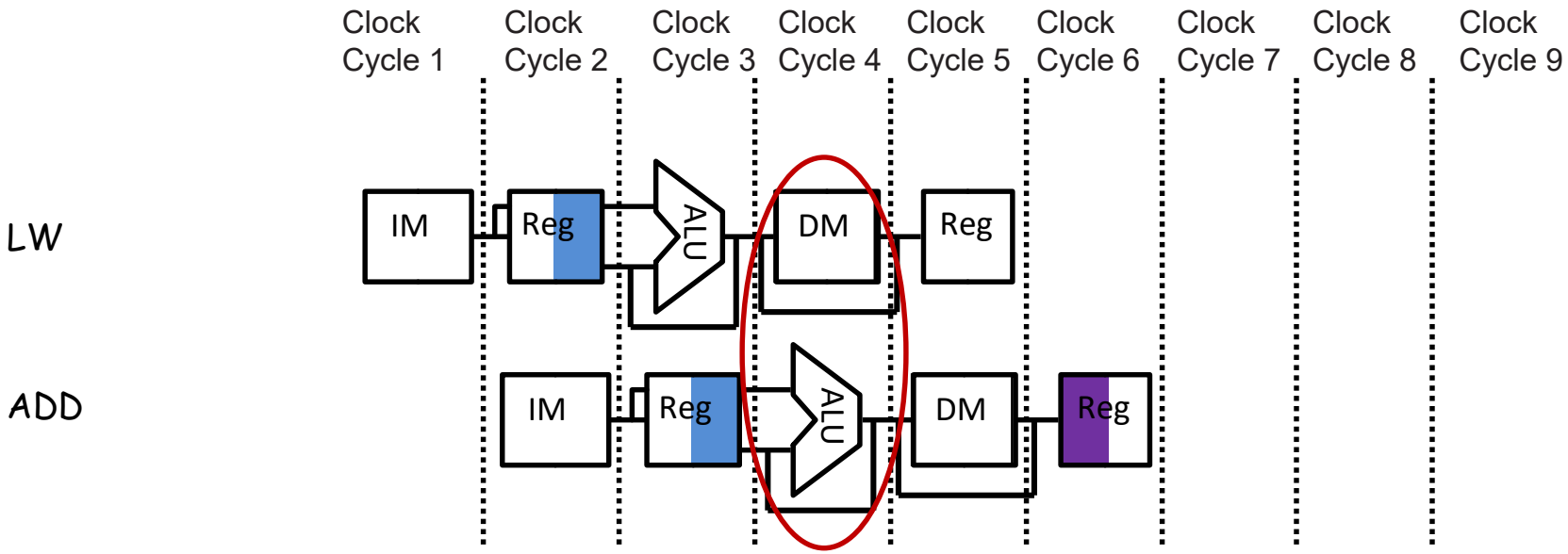
IF	取指时发生缺页；
	存储器访问边界未对齐；
	违反了存储器访问权限；
ID	未定义的或非法操作码；
EX	算术异常；
MEM	存取数据时缺页；
	存储器访问边界未对齐；
	违反了存储器访问权限；
WB	无

采用页式虚拟存储器

- 缺页（页面失效）时，表示程序需要的页不在内存，因此需要外部地址转换，再把磁盘上的页调入内存。
- 在操作系统中，把缺页当作一种异常故障来处理，它可能发生在一条指令的执行过程中，如取指、取操作数、存结果。

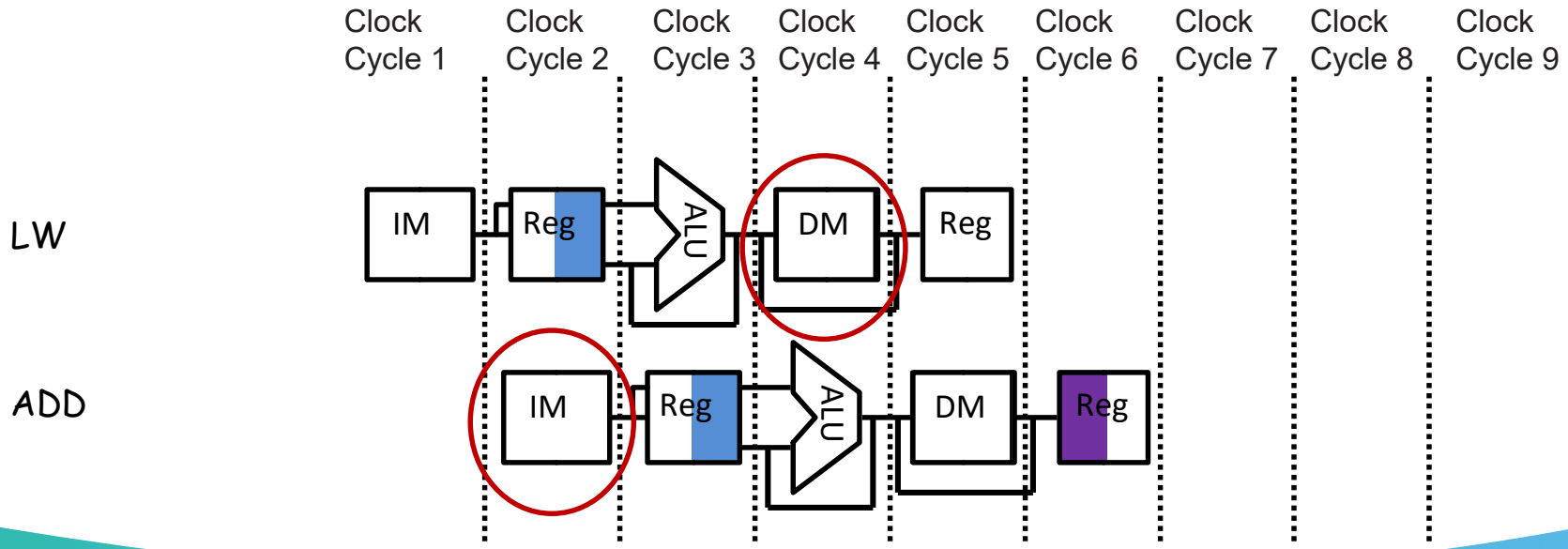
整数MIPS一个时钟周期中的多个异常

- 在Clock Cycle 4，LW指令可能出现数据缺页异常，且ADD指令可能出现算术异常。CPU应该先响应哪个异常?
- 应该先响应前一条指令，即LW指令的缺页异常，然后重新启动LW指令的执行。
- 当LW指令执行完毕后，ADD's 算术异常会再次发生。



整数MIPS多个异常的乱序产生

- ADD指令在取指阶段有产生了一个异常，LW指令在访存阶段产生了一个异常。
- 如果我们要实现精确异常，LW 异常必须先被处理
- 实现精确异常：设置硬件实现按指令顺序处理异常。



指令集引起的复杂问题（整型指令）——定义提交

- 当一条指令保证能够正常执行完成时，则称它为已提交。
  - 在MIPS整数流水线中，所有指令在MEM阶段结束时，都是已提交。对于MIPS，没有指令在MEM级之前更新状态。
  - 如果在指令提交之前不需要更新处理器的状态，则容易实现精确异常处理。
- 在大多数RISC系统中，每条指令只写一次结果。
  - 这就意味着在指令提交前任何时间被废除，都不会影响系统的状态。

什么使得流水线难以实现？

- 指令集引起的复杂问题，如整数指令
  - 很复杂的多周期指令是难以实现流水线操作的
  - 例子：
  - stringMov from 0x1234, to 0x4000, 0x1000 bytes
  - 如果发生异常
  - 这些指令在提交前可能已经改写多个存储单元
  - 从指令的哪个位置重新恢复？改写的存储单元怎么处理？

指令集引起的复杂问题（整型指令）——CISC

- 不过对于很多*CISC* 处理器，并不是这样的情况，如 *VAX*
  - 对于这些处理器，可以在该指令或前序指令提交前修改机器的状态。
  - 例如，“Add R1，-(R2)”，指令使用自减寻址方式（修改寄存器）由于异常发生而被中止，则处理器状态有可能被修改。
  - 这就导致了一个非精确异常，使得难于恢复到异常发生之前的处理器状态来重新执行指令。

小结

- 异常是设计一个现代CPU最难处理的问题之一
  - 容易出错
  - 实现需要大量的特殊逻辑电路
- 现代CPU机制中的复杂性使异常处理更困难
  - 深度流水线Deep pipes
  - 超标量 --大量并行执行的指令
  - 乱序执行
    - 异常的时间顺序≠ 异常发生指令所在的程序顺序