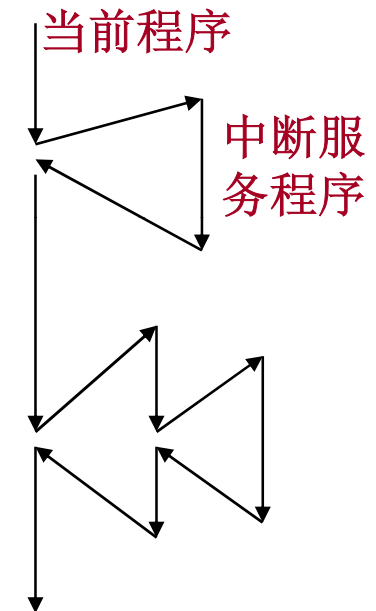


中断系统

llxx@ustc.edu.cn

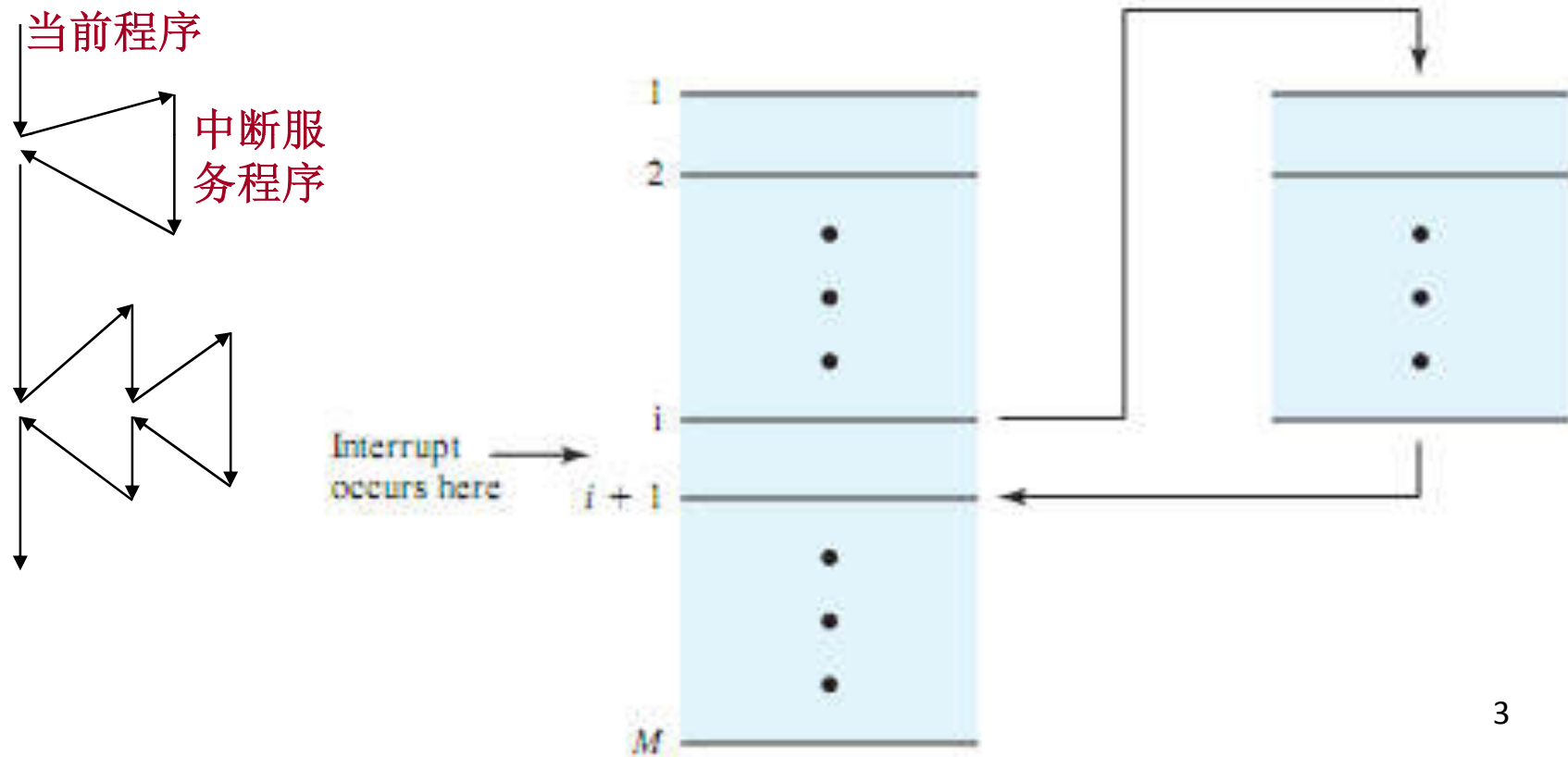
中断系统

- 中断的概念
 - 暂停当前程序的执行，转而执行其他程序。在它们执行完成后，恢复被中断程序的执行。
- 中断的作用
 - 异常：响应软硬件错误或故障，exc
 - I/O：intr
 - 系统与环境交互：实时响应外部事件（I/O，人机交互）
 - 设备间交互（多处理器(核)间通信）
 - 并发：提高计算机的整机效率
 - CPU与I/O并发
 - 多任务并发：允许单处理器“同时”执行多个任务
 - 服务：用户程序与OS间交互，trap
 - 一种提供系统服务和保护的机制
- 中断管理
 - 中断服务例程ISR
 - 中断机构



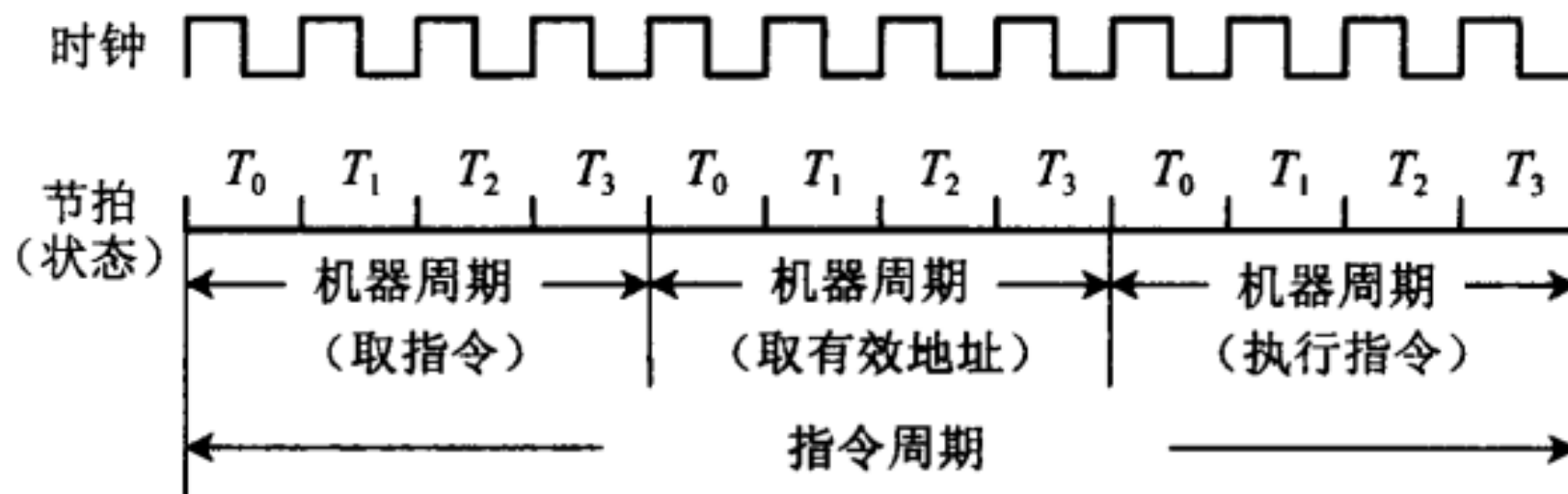
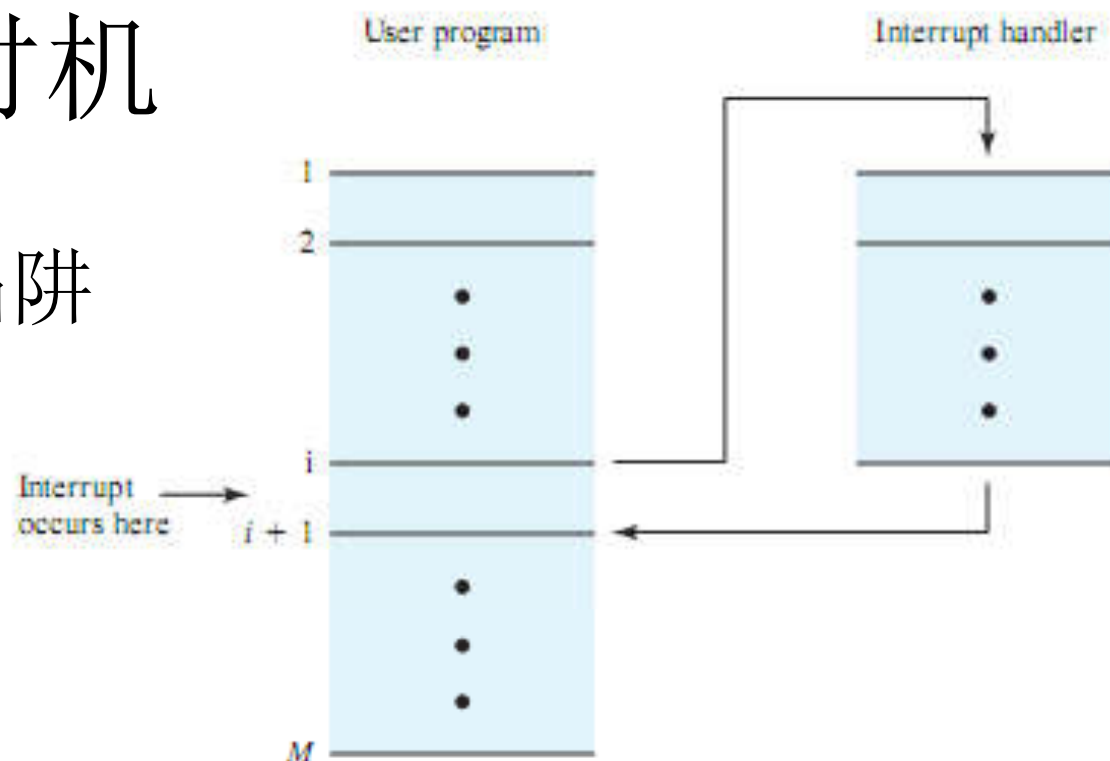
中断的行为

- Transfer of Control via Interrupts
- 中断嵌套



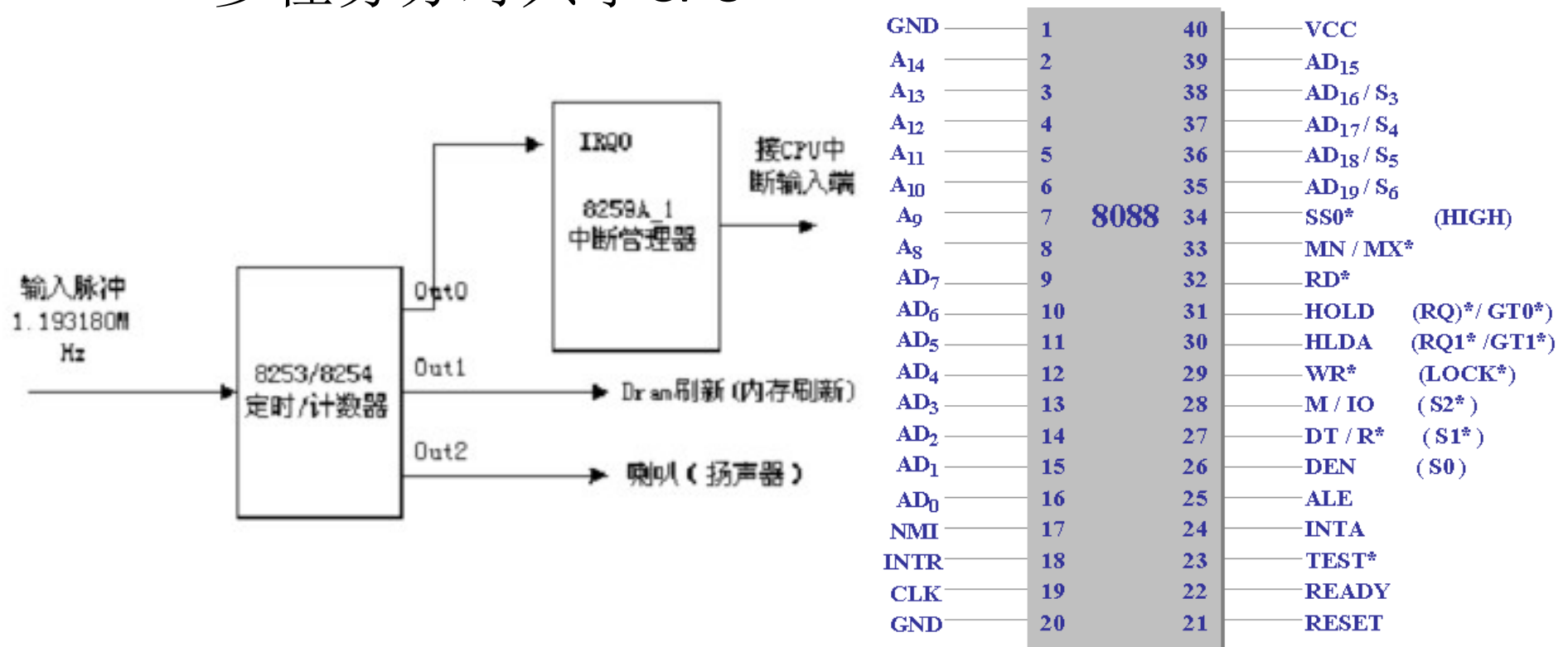
中断发生的时机

- 中断，异常，陷阱



例1：时钟中断（也称“软时钟tick”）

- 由可编程定时/计数器产生的INT
 - 维持系统时间(间隔大约10ms更新RTC)
 - 多任务分时共享CPU



例2： IBM PC的BIOS及DOS功能调用

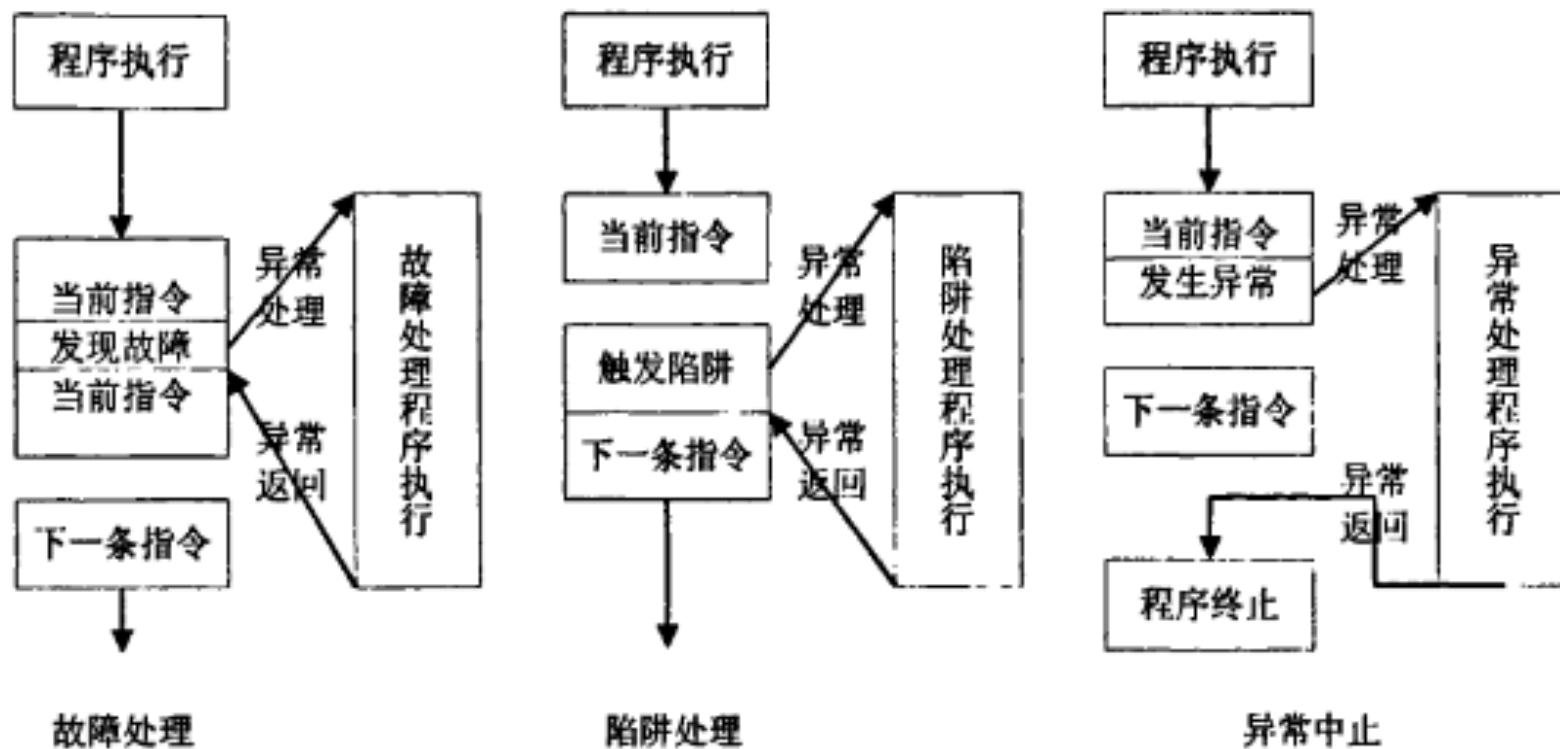
- 陷阱，系统调用

例： 字符输入输出

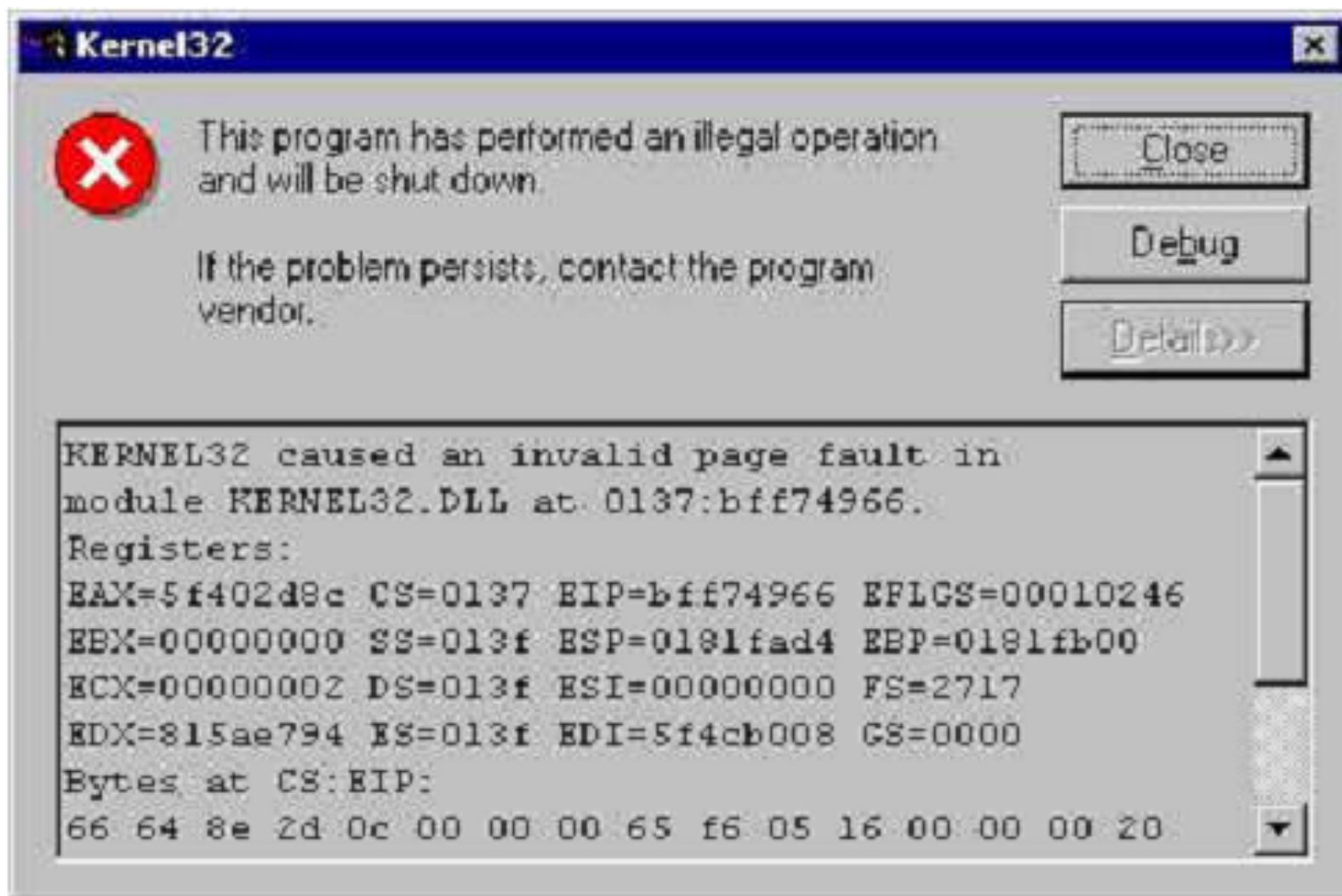
```
loop: mov ah, 0          ; 键盘功能调用 (int 16h)
      int 16h           ; al ← 按键的ASCII码
      mov bx, 0          ; 显示功能调用 (int 10h)
      mov ah, 0eh
      int 10h           ; 显示
getk:  mov ah, 0bh        ; 按任意键继续
      int 21h
      or al, al          ; al = 0?
      jz getk            ; al = 0, 没有按键, 继续等待
      jmp loop
      int 3h
```

异常处理模式

- 服务：将控制转移给OS的异常处理程序
 - 可恢复异常（下图中“故障”，OS处理后返回）
 - 不可恢复异常（异常中止，交用户处理）
- 同步：影响时钟周期宽度的确定



不可恢复异常：异常终止



Terms: exceptions、interrupts、traps

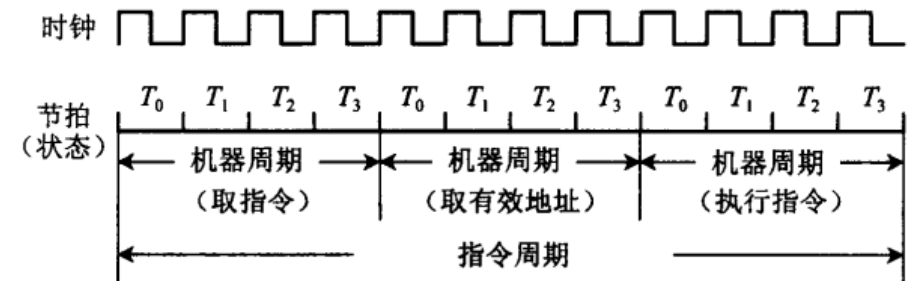
- x86: interrupt, 含
 - 外部中断: 硬中断 (hardware interrupt)
 - 可屏蔽中断, 不可屏蔽中断NMI
 - 内部中断: 软中断 (software interrupt)
 - 程序异常, 系统调用 INT n, 指令断点 (int 3调试)
- RISC: exceptions, 含
 - 外部事件External events(interrupts): I/O中断
 - 异常Exceptions: 软(硬)件故障
 - 陷阱Traps: syscall, 断点break, 自陷TEQ

中断管理需要解决的问题

1. 中断源如何向CPU提出中断请求（软/硬中断）
2. 多个中断请求时如何确定优先响应顺序
3. CPU响应中断的条件、时间点、方式
4. 响应中断后如何保护现场
5. 如何转入中断服务程序执行
6. 如何恢复现场，如何返回断点处执行
7. 中断处理过程中出现新的中断请求如何处理

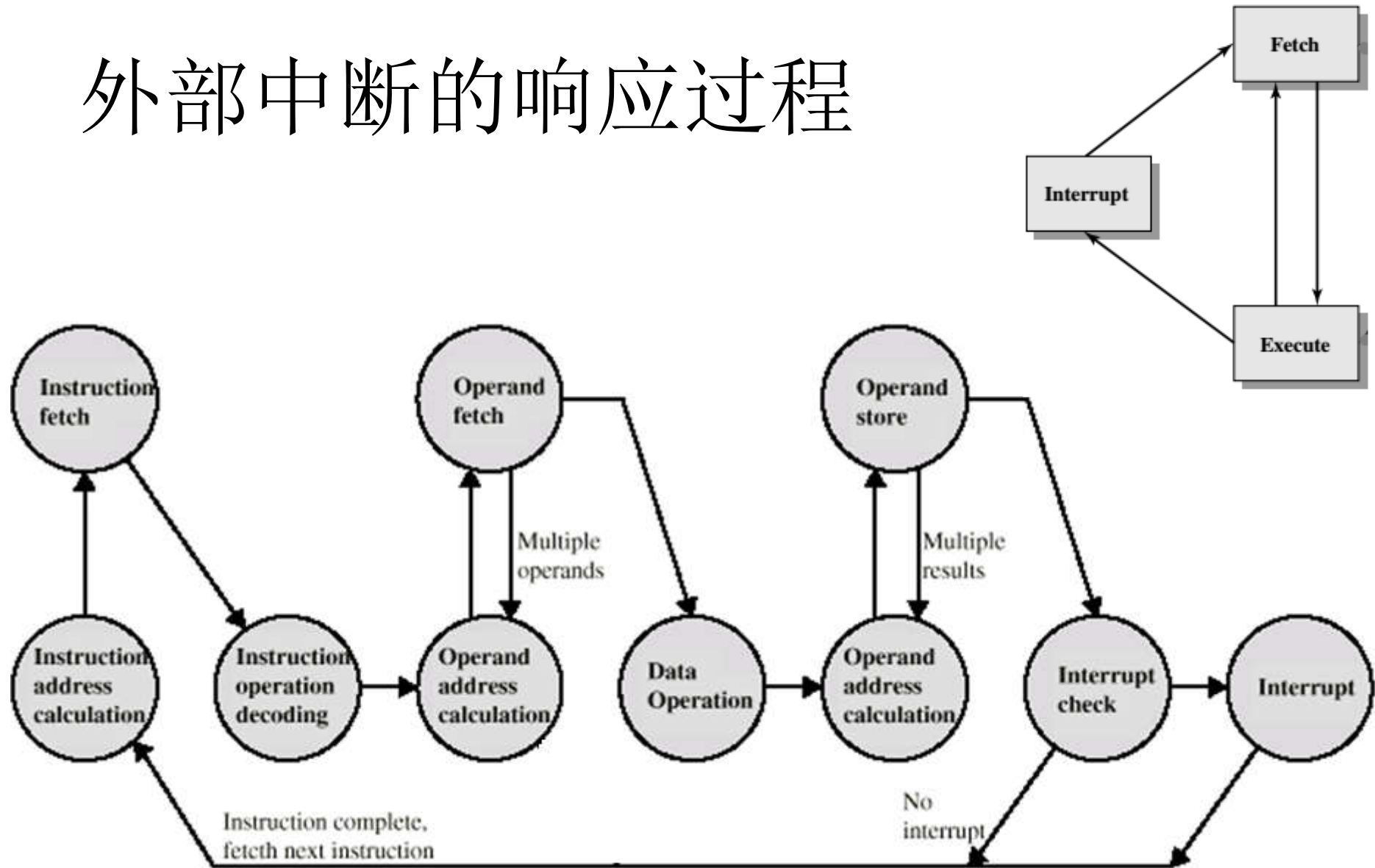
中断发生与CPU响应的时机与条件

- 断点、状态、现场（上下文）、返回点
 - 为了保证程序执行的顺序性, 没有执行完的指令必须记录它们执行到哪一个阶段
 - PC/nPC, PSW
 - 上下文（通用寄存器）



- 中断发生与CPU响应时间
 - 外部中断：异步（延迟响应）
 - 指令周期结束（顺序执行模型！）
 - 内部中断：同步（“马上”响应）
- 响应中断的条件
 - 外部中断：
 - 可禁止（PSW的中断允许位），可屏蔽（中断屏蔽寄存器）
 - 内部中断：不可禁止

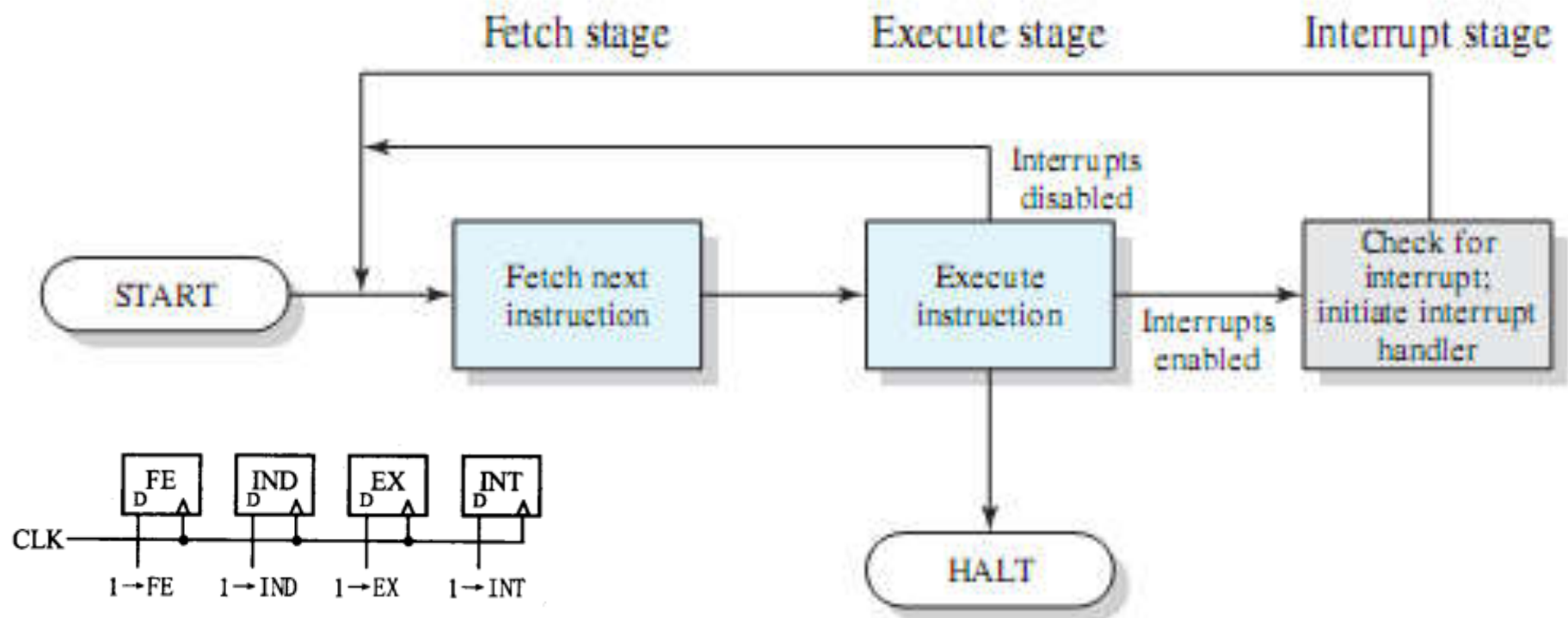
外部中断的响应过程



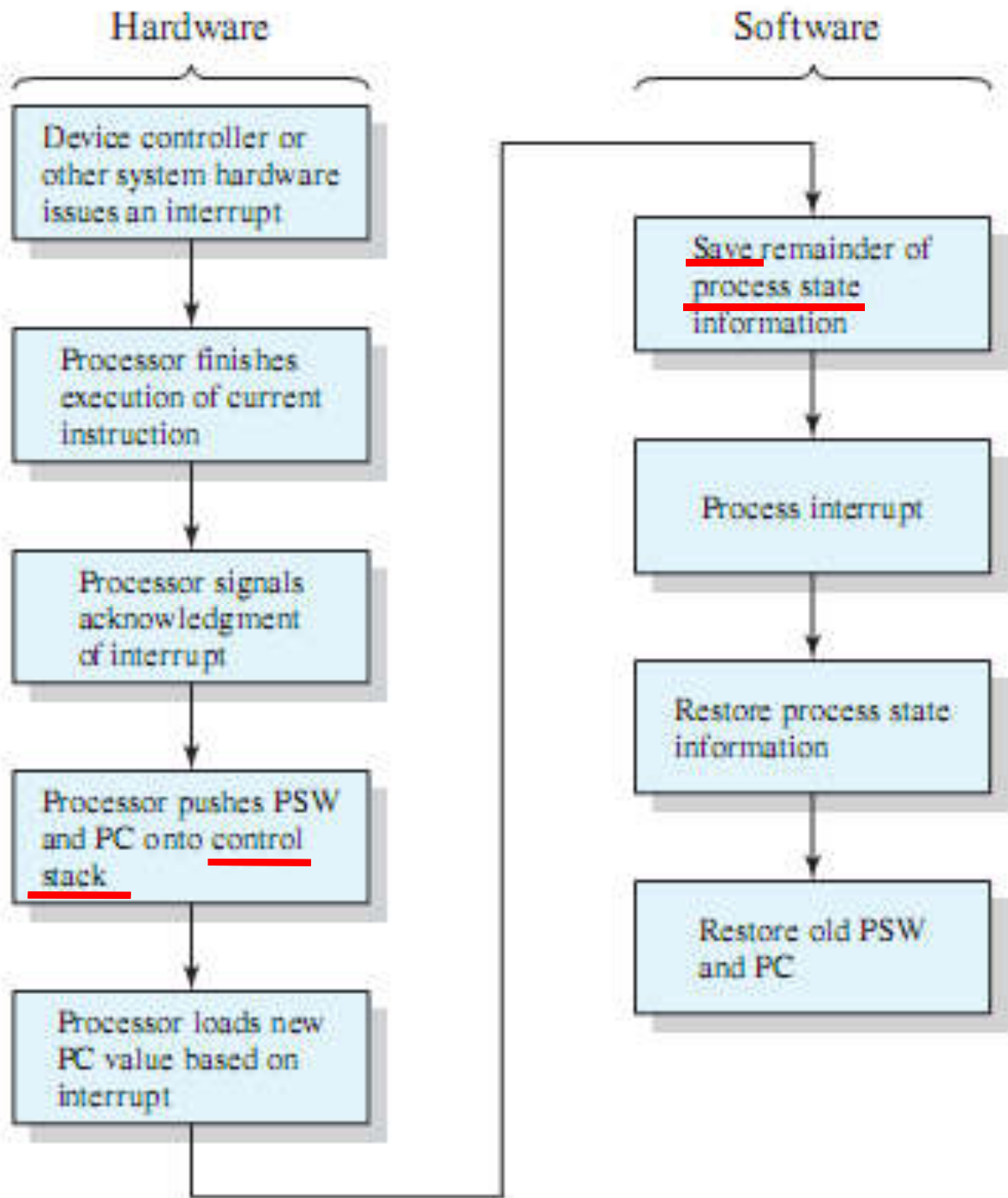
Instruction Cycle (with Interrupts)

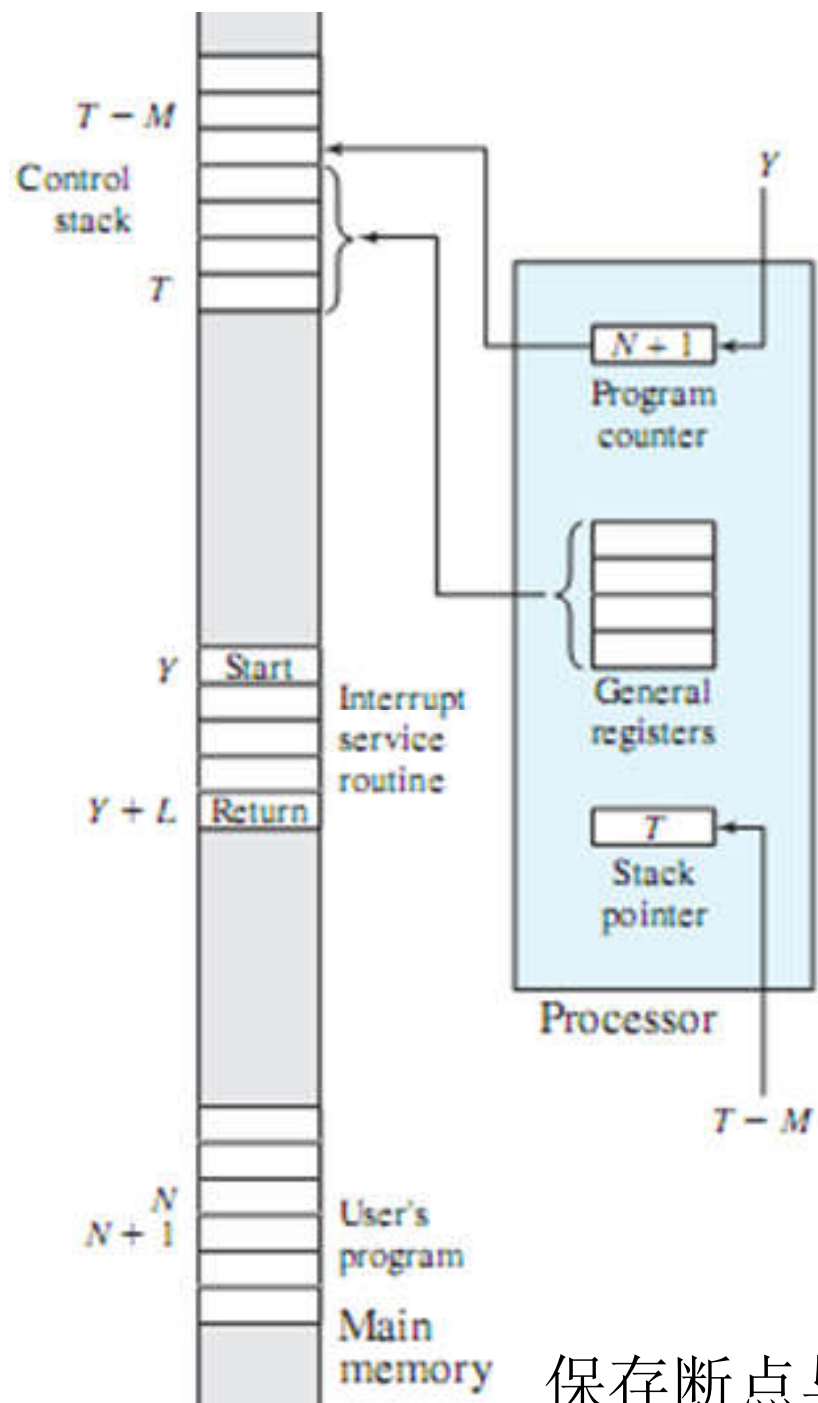
中断周期(Instruction Cycle with Interrupts)

- CPU与中断控制器通信，响应外部事件
 - 有中断请求？识别中断源，获得中断向量
- 动作：存PC和PSW，关中断，识别，转ISR

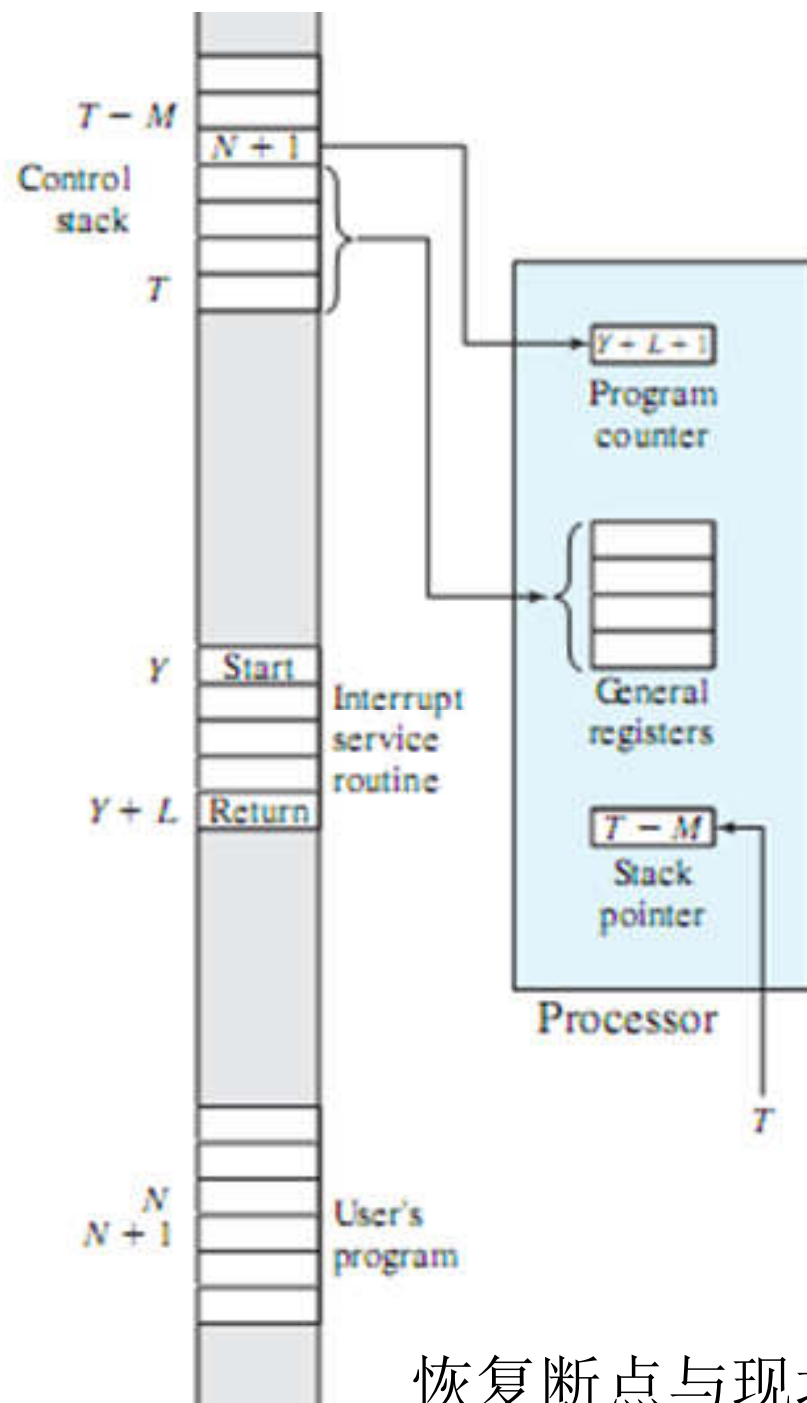


Simple Interrupt Processing





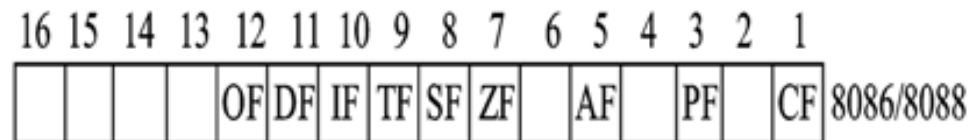
保存断点与现场

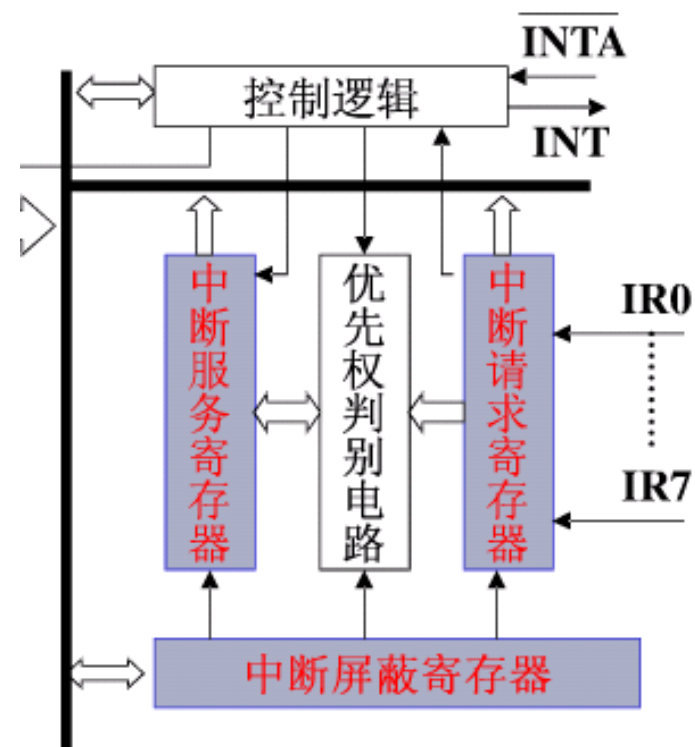
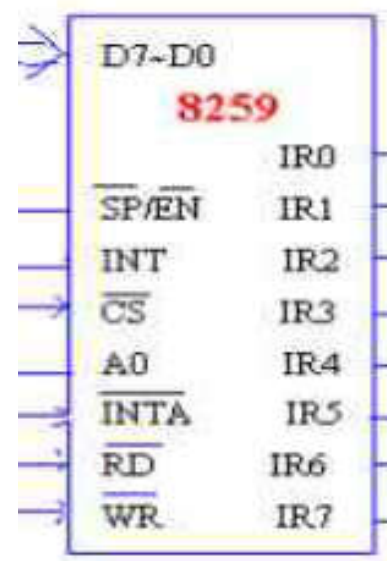
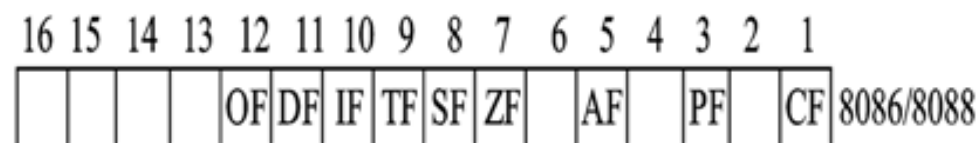
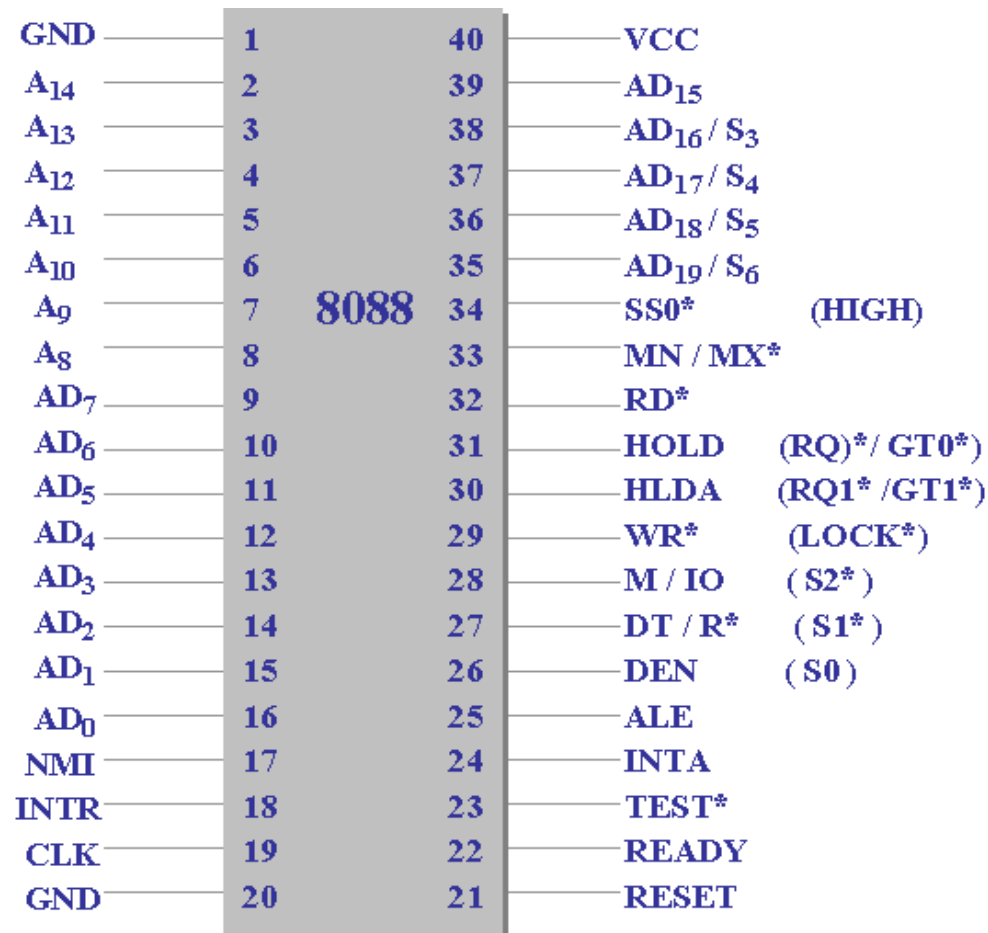


恢复断点与现场

中断机构组成

1. CPU中断禁止/允许控制：IF@PSW
2. CPU中断请求/响应控制：INTR、INTA
3. 中断响应/返回：中断隐指令
4. 断点/现场保存：MEM（stack）
5. 中断服务
 - 中断源识别/判优：中断控制器
 - ISR入口：向量方式、非向量方式





MIPS中的异常

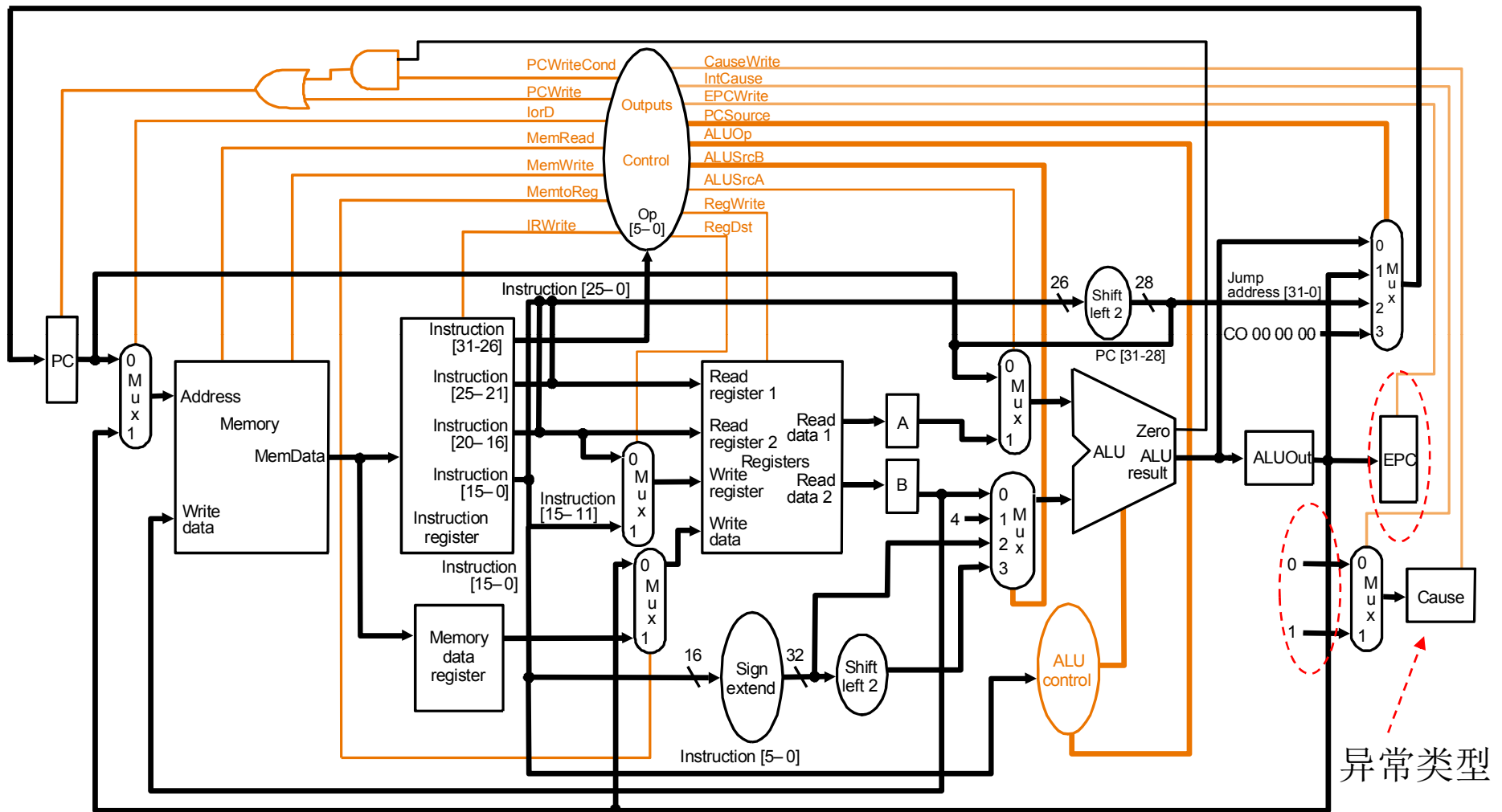
- External events
 - 中断或读总线错等
- Memory translation exceptions
 - 缺页或越界等
- Other unusual program conditions for the kernel to fix
 - 须内核处理的不常见程序状态
- Program or hardware-detected errors
 - 非法指令、溢出、对齐等
- Data integrity problems (Checksum etc.)

Type of event	From where?	MIPS terminology
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Arithmetic overflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

MIPS的异常（软中断）处理


- 设计控制部件的难点在于异常处理
 - 检查异常和采取相关的动作通常在关键路径上进行
- 讨论两种异常：非法指令和算术溢出
- 异常处理
 - 断点保存：将受干扰的指令的地址保存在EPC寄存器中
 - 异常识别：根据状态寄存器cause中的异常原因分别处理异常
 - 非法指令：为用户程序提供某些服务
 - 溢出：对溢出进行响应
 - 异常服务程序：停止异常程序的执行并报告错误等
 - 异常处理程序地址在c0000000H

多周期模式的异常处理



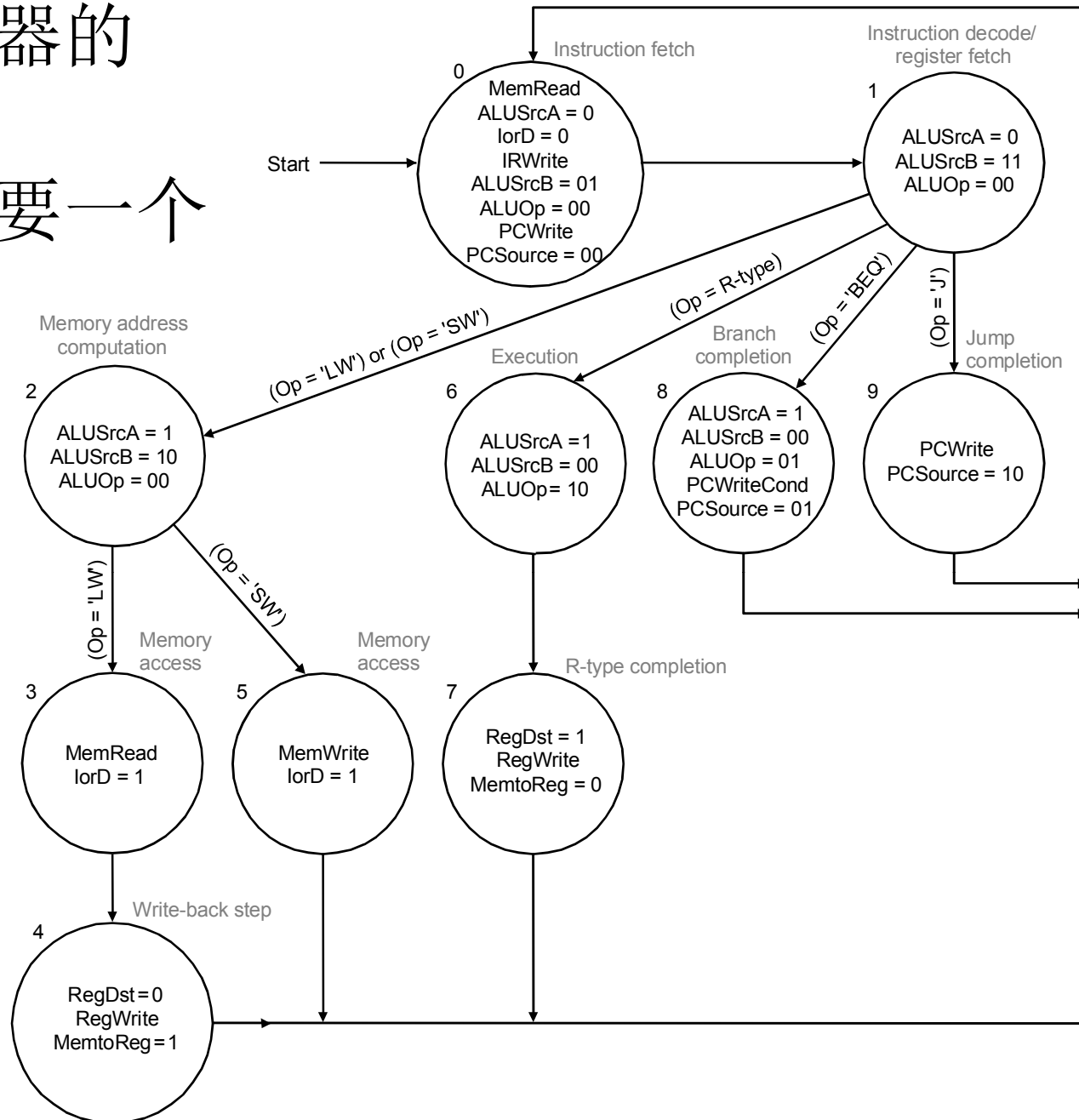
Exceptions (overflow)

Step	R-Type	lw/sw	beq/bne	j	Exception
IF	IR = Mem[PC] PC = PC + 4				
ID	A = Reg[IR[25-21]] B = Reg[IR[20-16]] ALUOut = PC + (SE(IR[15-0]) << 2)				
EX	ALUOut = A op B	ALUOut = A + SE(IR[15-0])	If (A==B) then PC = ALUOut	PC = PC[31-28] (IR[25-0] << 2)	PC = 0x8000 0180 EPC = PC - 4 Cause = 0 or 1 Status = ?
MEM	Reg[IR[15-11]] = ALUOut	MDR = Mem[ALUOut] Mem[ALUOut] = B			
WB		Reg[IR[20-16]] = MDR			

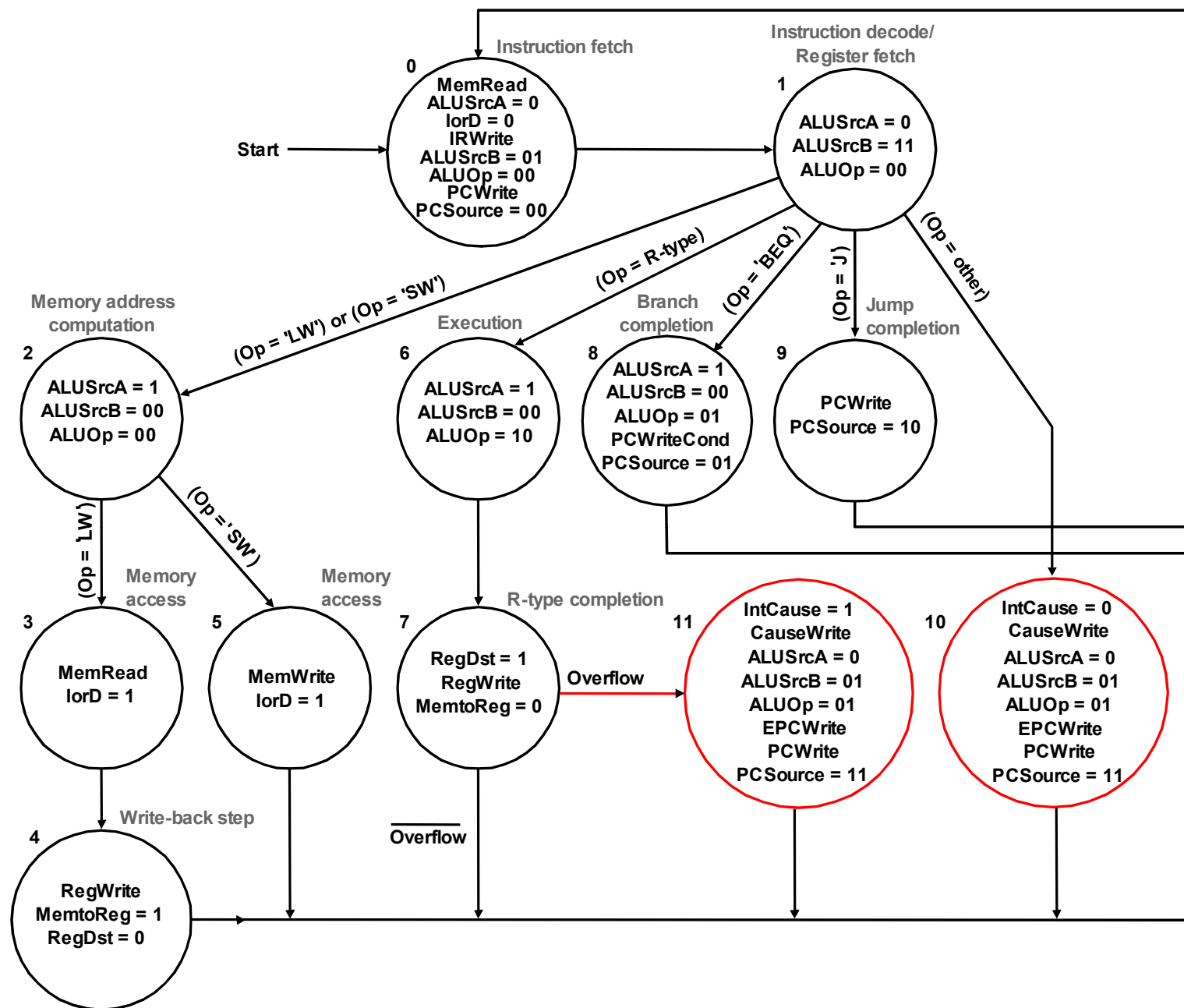


- The exception steps are done in one clock cycle.

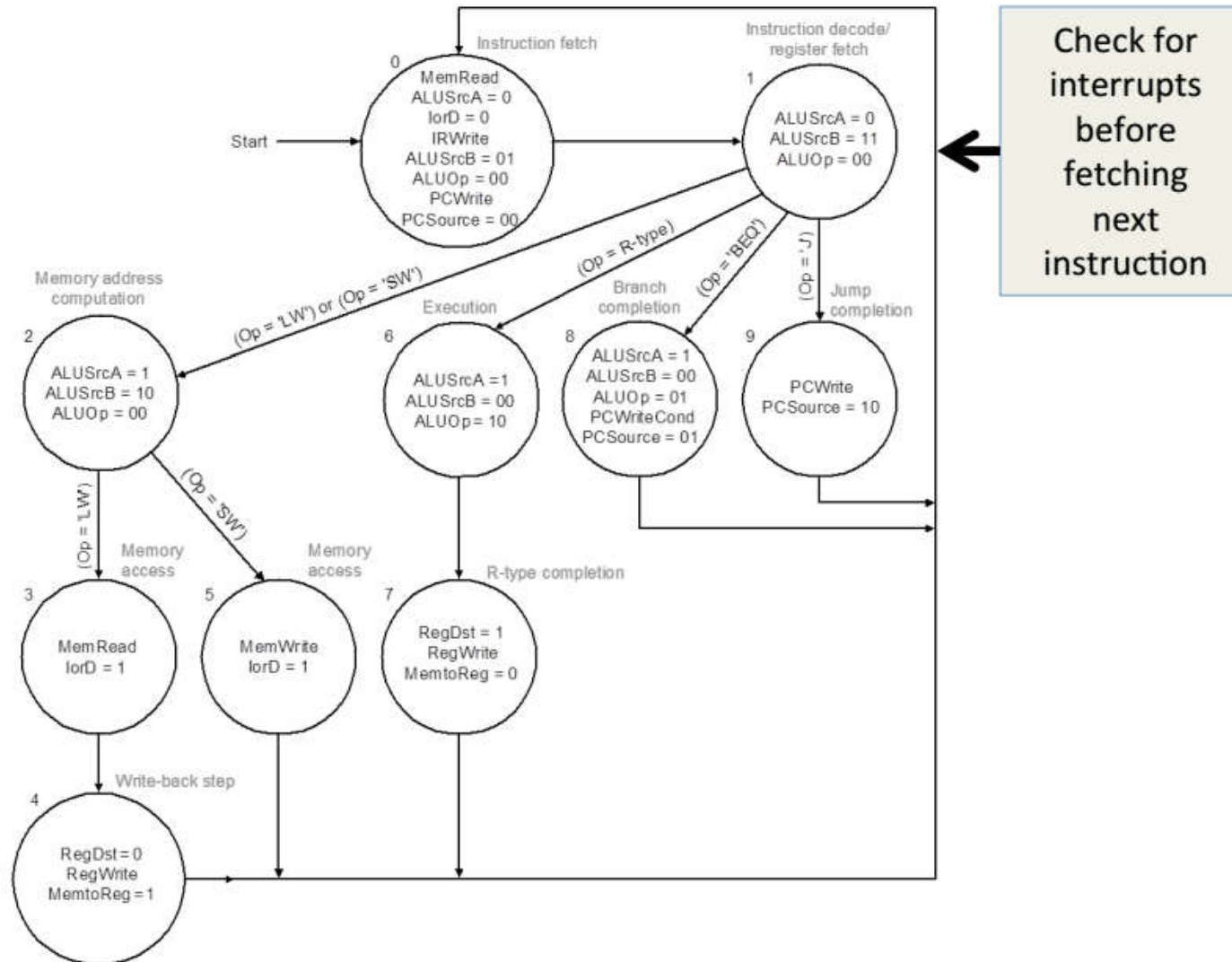
多周期控制器的 MooreFSM， 每个状态需要一个 时钟周期。



异常处理控制

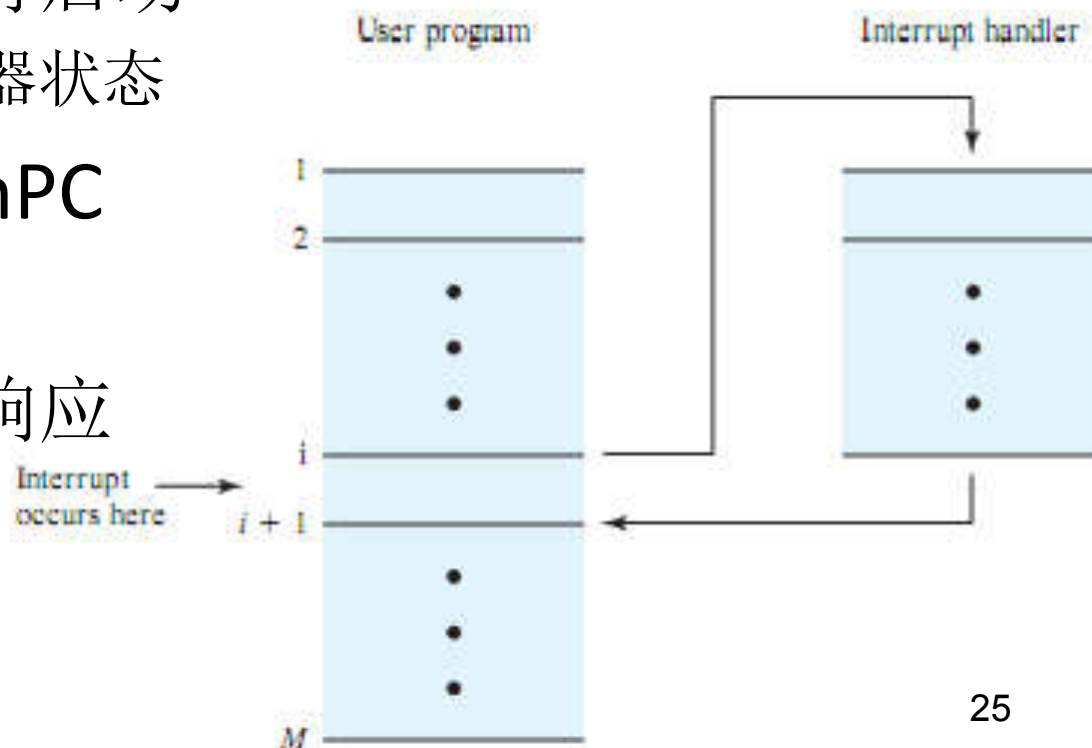


Multicycle Interrupts



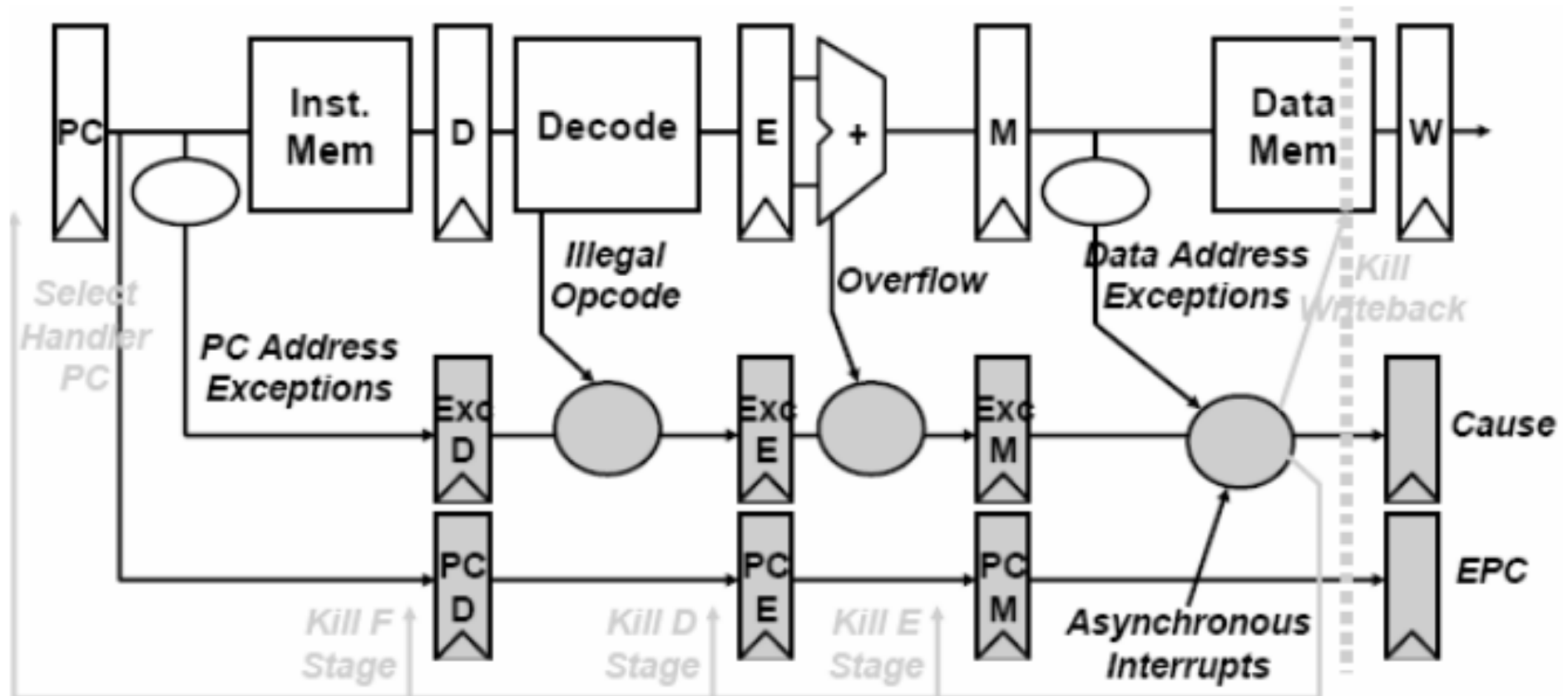
非流水线的中断和异常语义

- 顺序语义模型
 - 之前的指令都已执行完成
 - 已经提交其状态
 - 之后的指令还没有启动
 - 没有改变任何机器状态
- 断点精确：= PC/nPC
 - 中断：异步响应
 - 指令异常：同步响应
 - 包括软中断
- 现场简明



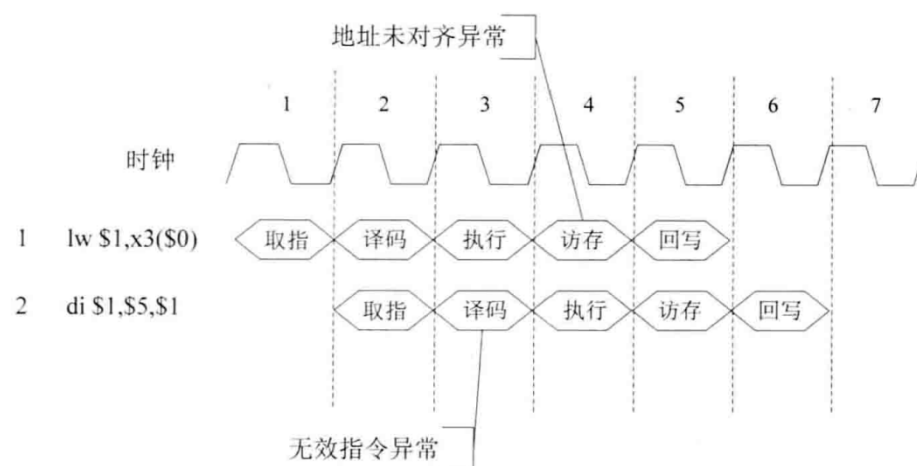
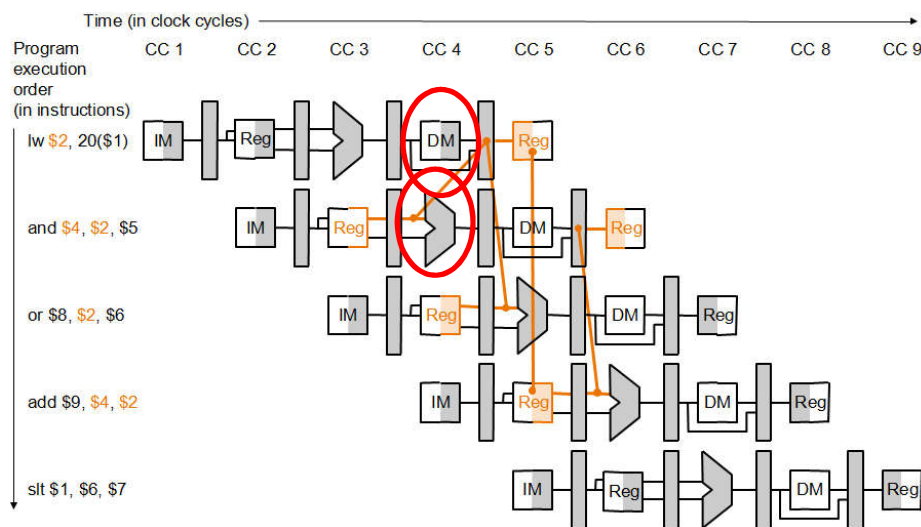
流水线中的异常与中断

- 多个流水段，多条指令，多种异常并发
 - 异常：访存、译码、溢出
 - 中断



流水线异常处理的挑战

- “顺序执行”只是一种抽象：断点和状态难以确定
 - 转移预测失误可能取消其后出现了异常的指令
 - 后续指令在产生异常指令完成之前改变了系统的部分状态
 - 如 `ld` 指令在 **MEM** 段产生异常，而其后的 **R-type** 指令设置了 **O** 标志
 - 异常发生顺序与指令执行顺序不一定相同
 - 中断？



方案1：非精确异常

- 语义

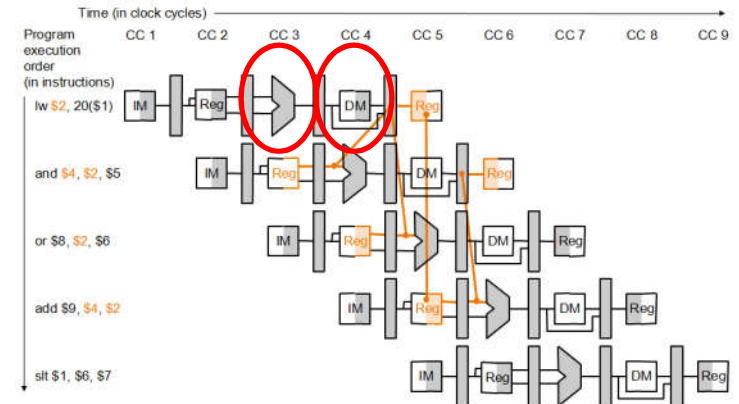
- 允许已进入流水线中的指令**执行完**再转去执行中断处理

- 无论在第*i*条指令的哪一流水段上发生异常，都不再允许后继指令进入流水线

- 断点：最后进入流水线的那条指令的地址

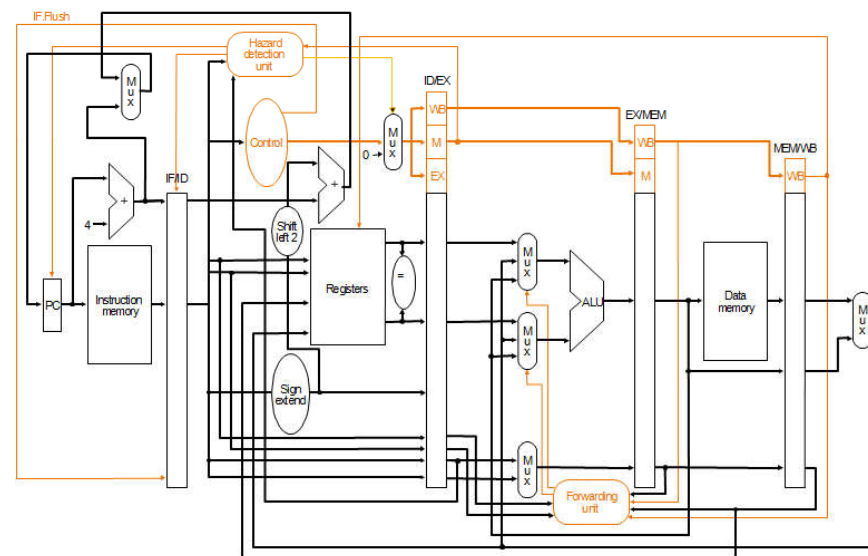
- **非精确**（不是“当前指令”），**且可变**（不同段发生异常，EPC**增量**不同）

- 优点：硬件比较简单

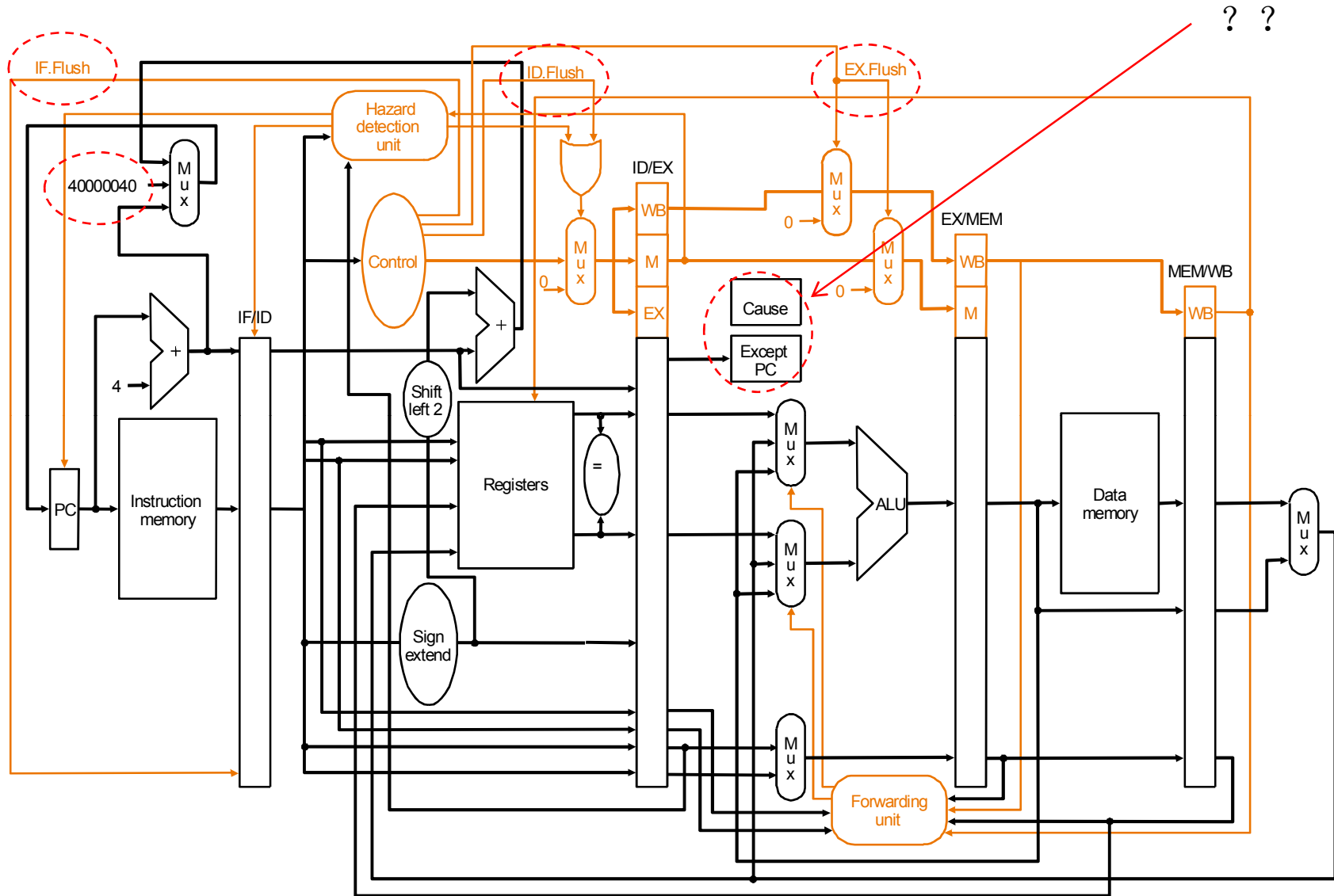


DLX实现非精确

- 将异常视为一种控制依赖
 - 暂停指令流中导致异常的指令
 - 执行完之前的所有指令
 - 清除之后的所有指令
 - 记录异常原因
 - 保存**最后进入**流水线的指令的地址（断点）
 - 转异常处理程序
- 不允许后续指令继续执行
 - 与“非精确语义”不一致！

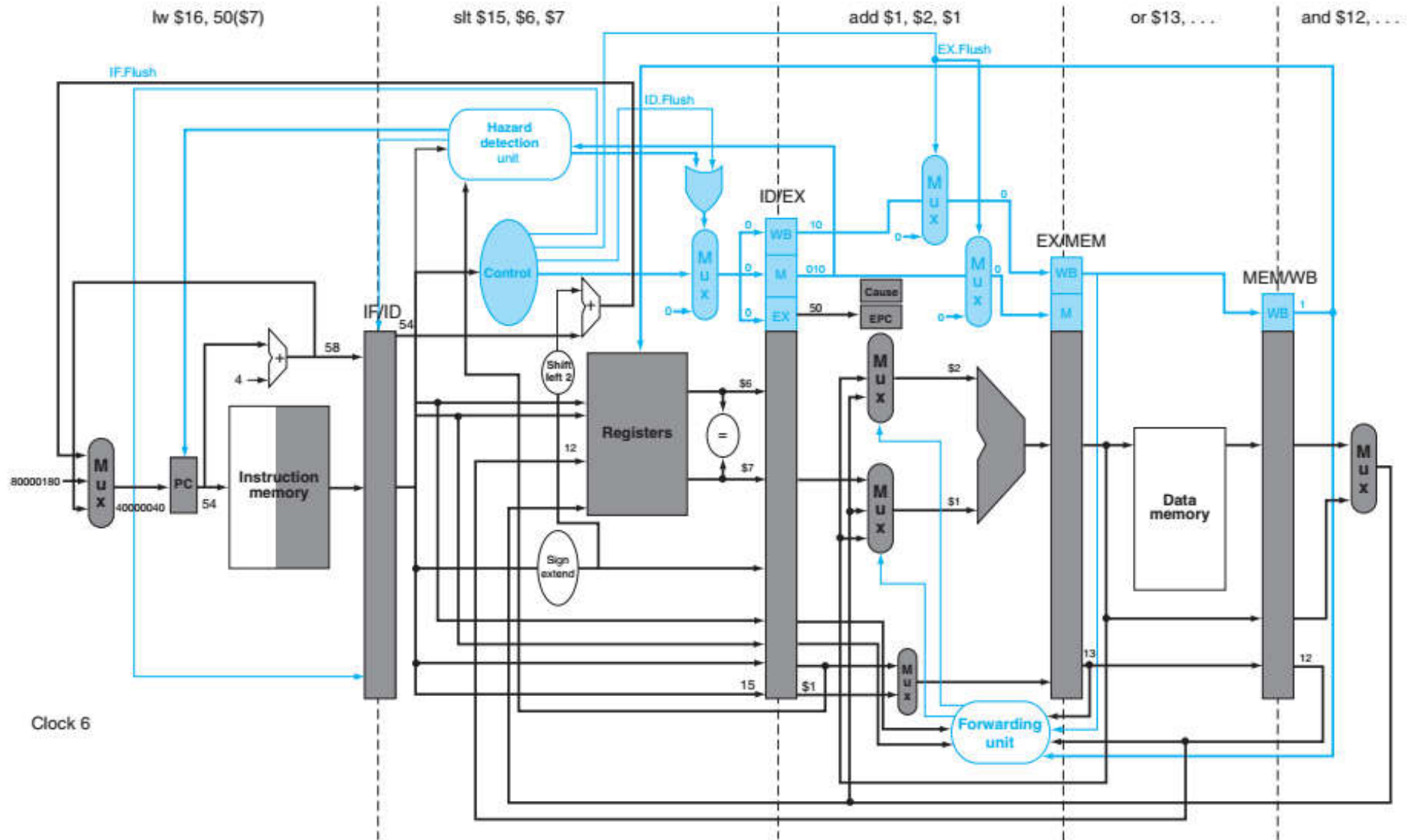


非精确异常控制：flush

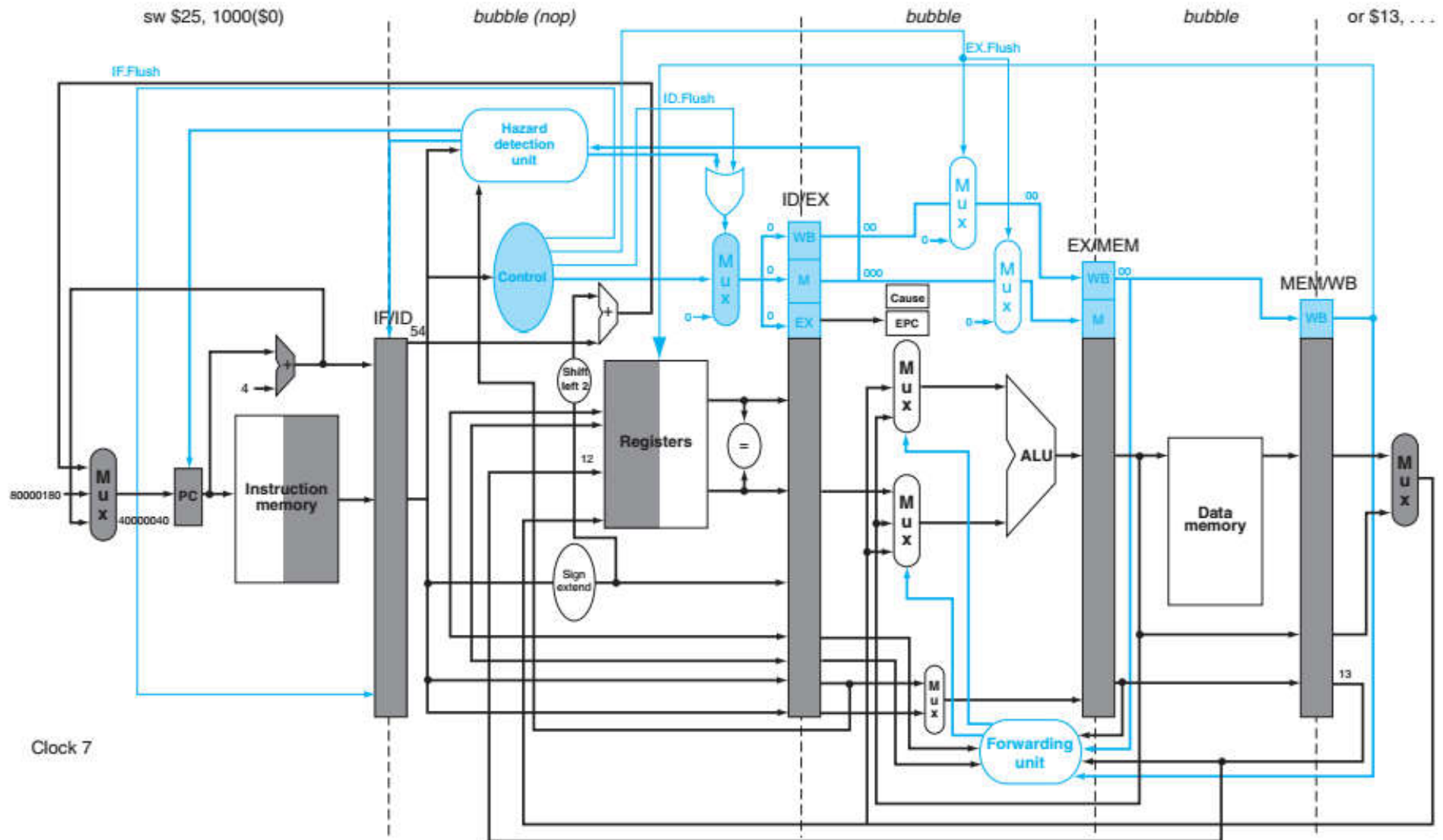


COD4 图 4-67

40 _{hex}	sub	\$11, \$2, \$4
44 _{hex}	and	\$12, \$2, \$5
48 _{hex}	or	\$13, \$2, \$6
4C _{hex}	add	\$1, \$2, \$1
50 _{hex}	slt	\$15, \$6, \$7
54 _{hex}	lw	\$16, 50(\$7)
...		



COD4 图 4-67



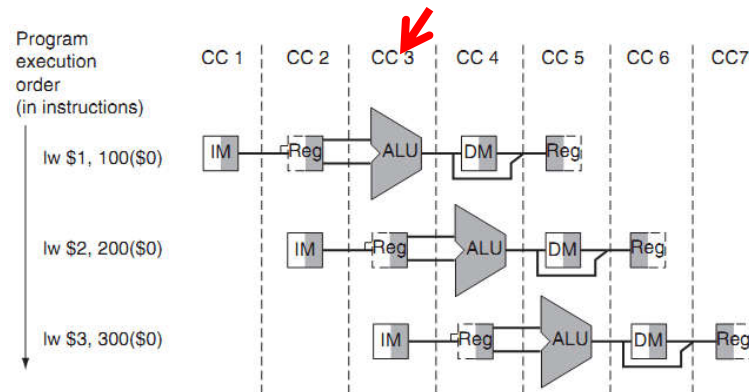
非精确异常的问题

- 缺点

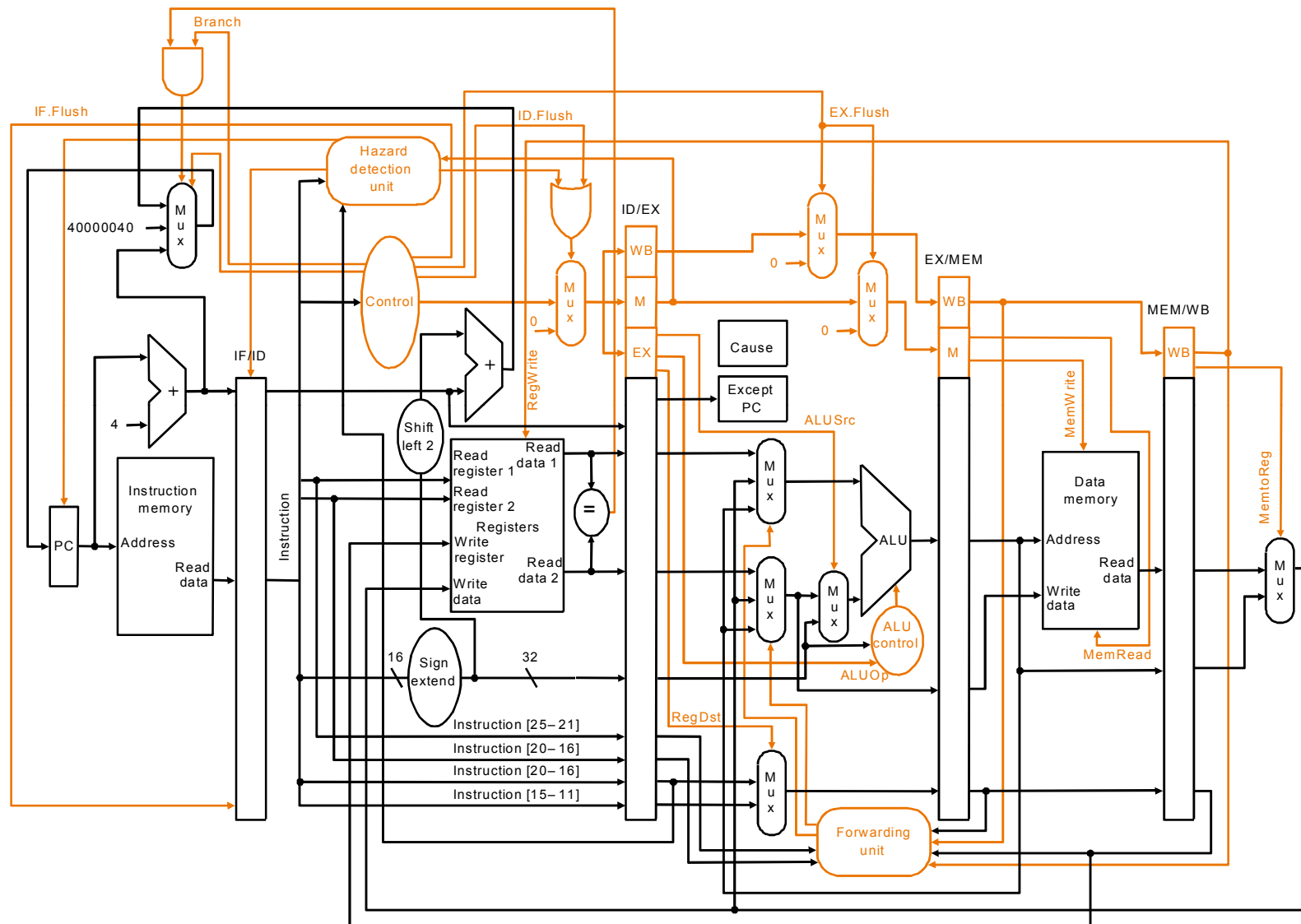
- 异常响应时间较长（抖动）：不同段异常时
- 可能会导致程序出错：异常
 - i: FADD R1, R2; $(R1)+(R2) \rightarrow R1$, 如果此时溢出？
 - i+1: FMUL R3, R1; $(R3) \times (R1) \rightarrow R3$, 无效执行！
 - （此处假设有forwarding）
- 程序调试不便：
 - 程序员在第 i 条指令设置断点, 但程序不能准确中断在所设置的断点处。

- 中断？

- 多个异常？



The final datapath & control

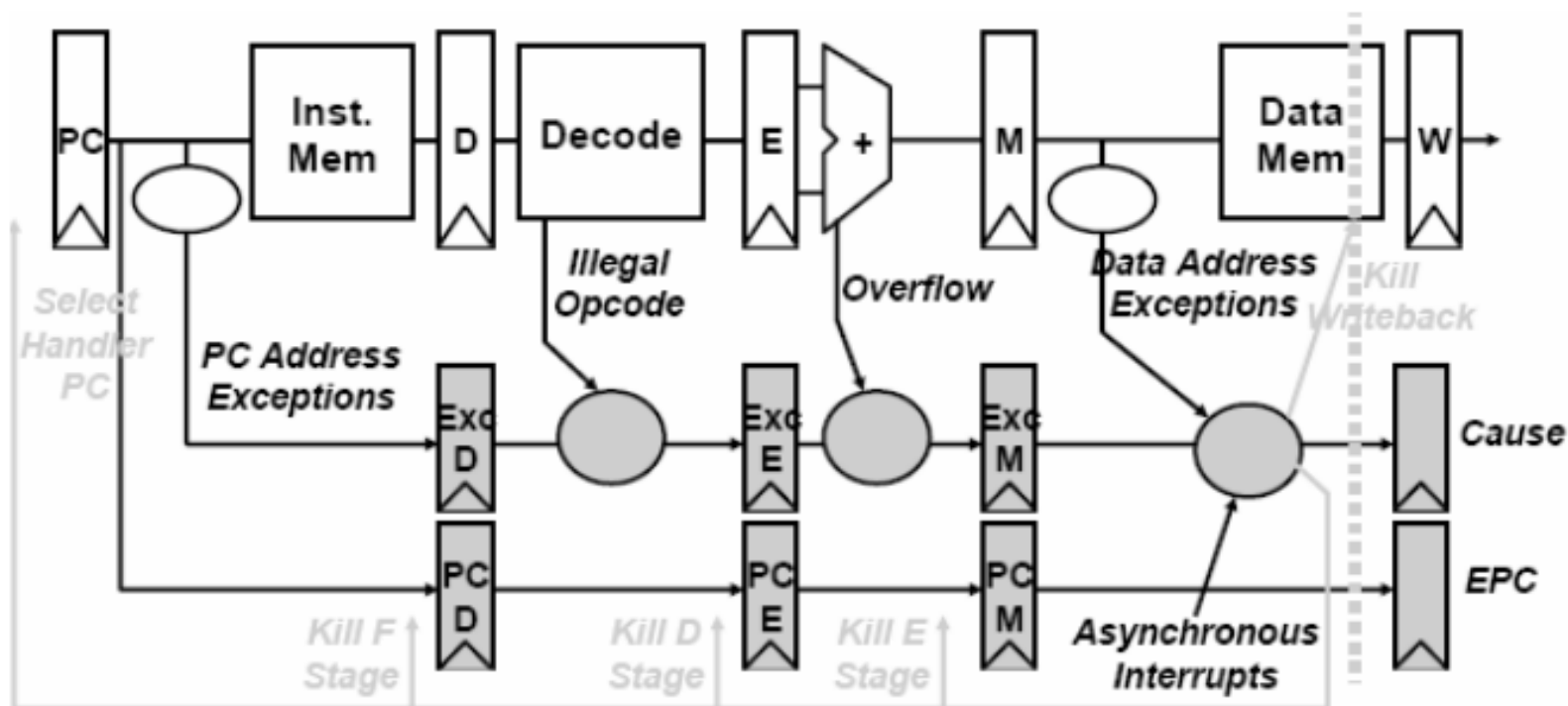


方案2: MIPS实现精确异常

- 实现“精确”开销大
 - 要保证异常指令“可重启”
 - 安全停止流水线，并完整保存当前状态
 - 需要大量后援寄存器保存流水线中各指令的现场
 - 包括RegFile和流水段寄存器（含各段的控制寄存器）！
- MIPS采用提交点技术实现精确异常，简化了设计
 - 提交点：M段（可否其他段？）
 - 多个异常：先发生的异常并不立即处理，只是被标记
 - EXCn寄存器：保存异常类型
 - 流水线中最深的指令引起的异常最优先
 - 中断：在M段检测（符合“异步”语义）
 - 断点：EPC = PC or nPC?
 - 中断断点 = MEM段的当前指令（返回后要重新执行！）

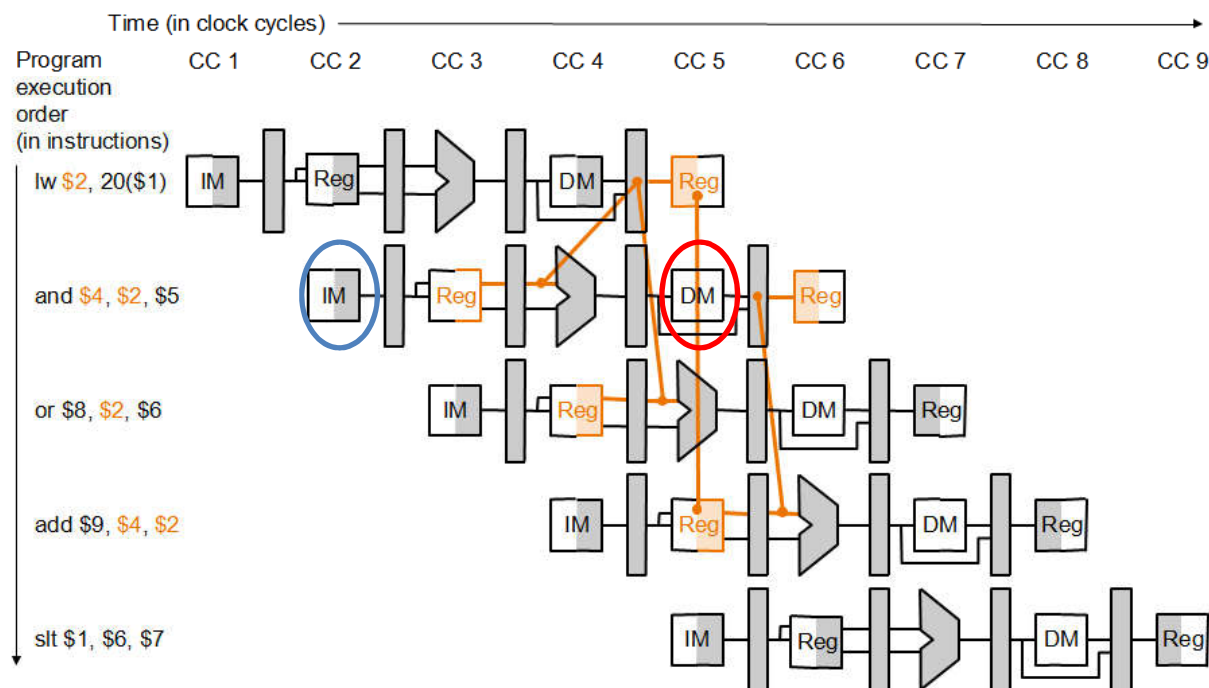
各段产生的异常及处理：MIPS的策略

- 保持流水线的异常标记直到**提交点**（M段）
- 早期流水段的异常抑制后来的异常（flush IF/ID/EX）
- 提交点处引入**外部**异常（**抑制**其他异常）
- 如果提交点有异常，则更新**cause**和**EPC**，清除所有流水段，回复**PC**值到**fetch**段

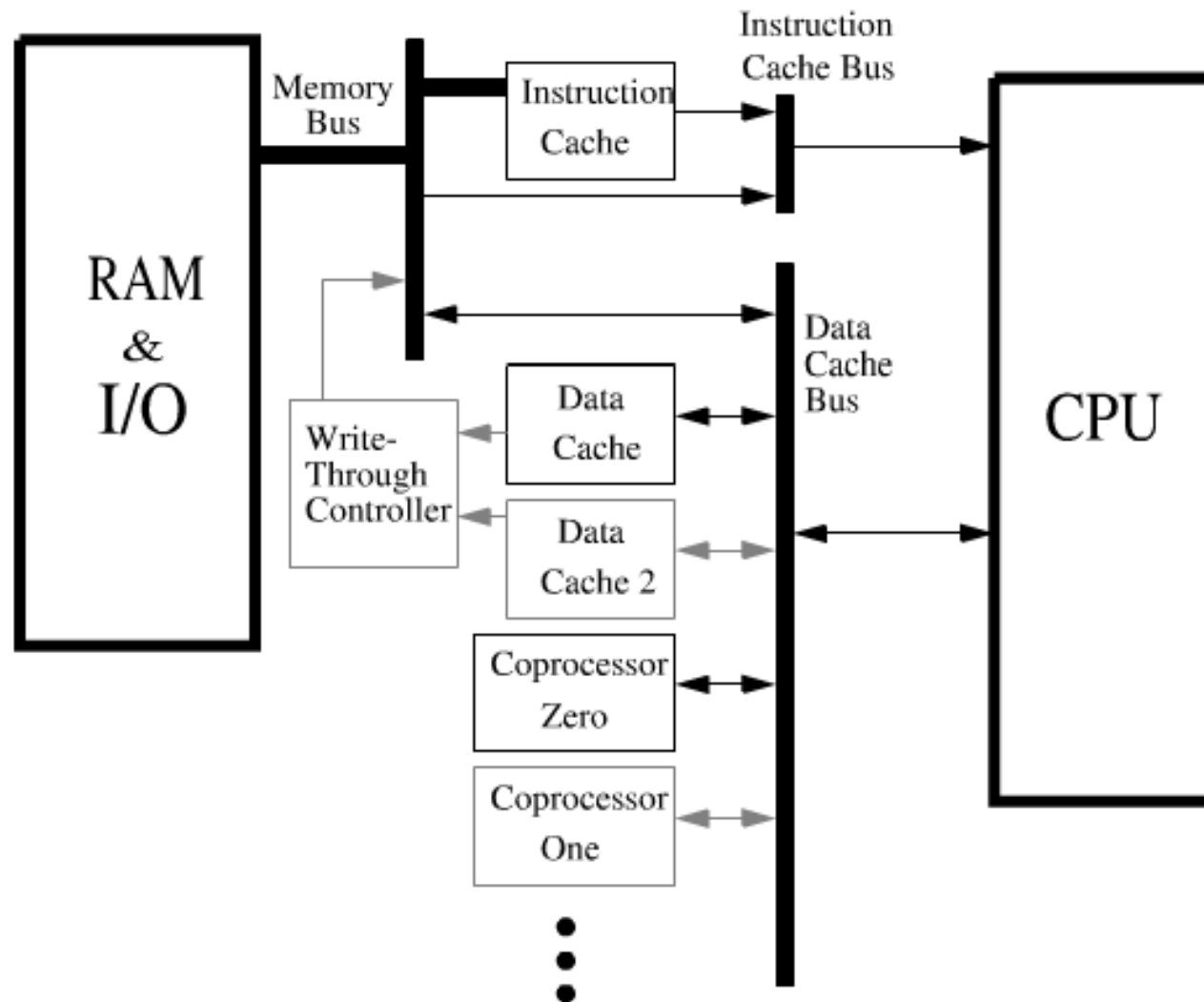


为何是M? EXE或WB也OK?

- 例：如果and出现取指缺页错，直到MEM才处理
 - 前一条指令已执行完成
 - 后续指令被flush（kill），保证没有指令完成写回
 - 异常返回重新执行and

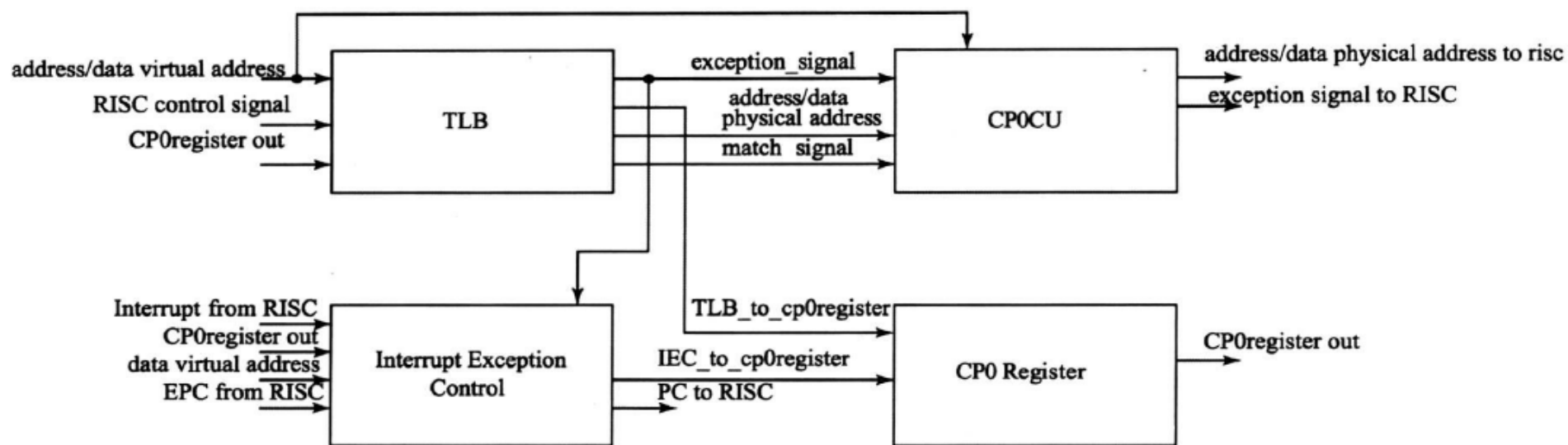


MIPS R3000 & Coprocessors



MIPS CP0

- 控制CPU
 - 异常处理、MMU

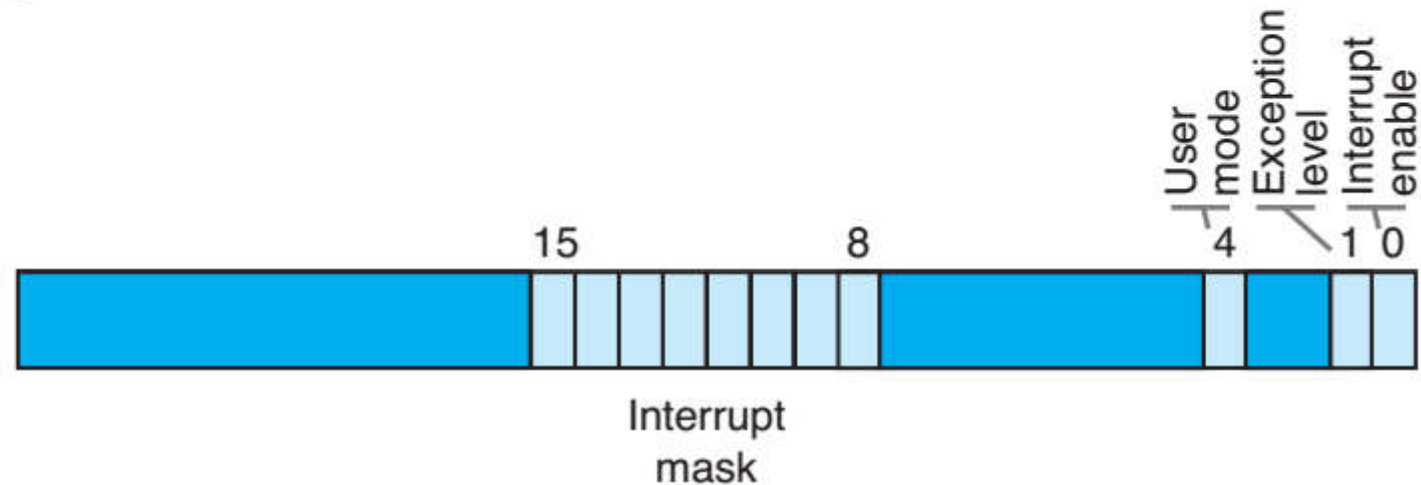


MIPS CP0 Reg

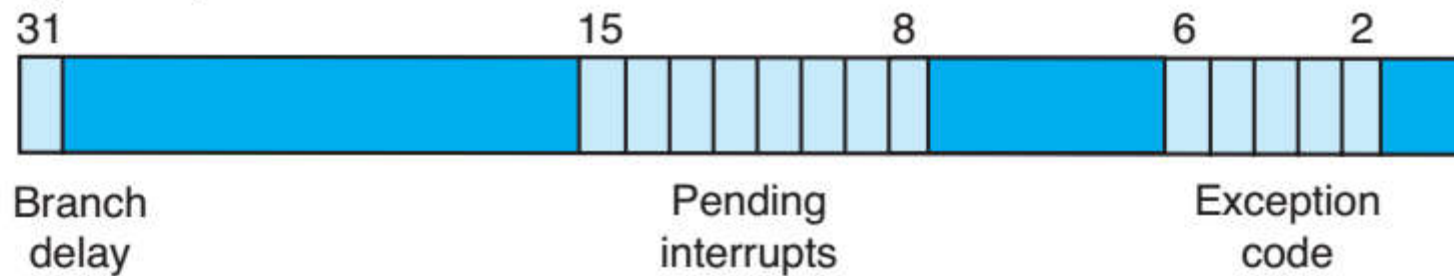
- 含32个寄存器，用于控制CPU的工作模式
 - Register 0: Index，作为MMU的索引
 - Register 4, Context，用以加速TLB Miss异常的处理
 - Register 8, BadVAddr，发生TLB Miss或Address Error时的地址
 - Register 9, Count，计数频率是系统主频的1/2
 - Register 11, Compare，Comp和Count相等则触发硬中断IP7
 - Register 12, Status，特权级，中断屏蔽，中断允许等。
 - Register 13, Cause，异常的原因(interrupt pending)。
 - Register 14, EPC，异常发生时系统正在执行的指令的地址
 - Register 18/19, WatchLo/WatchHi，用于设置硬件数据断点
 - Register 28/29, TagLo和TagHi，用于高速缓存(Cache)管理。

Important Exceptions registers

Status register \$12:



Cause register \$13:

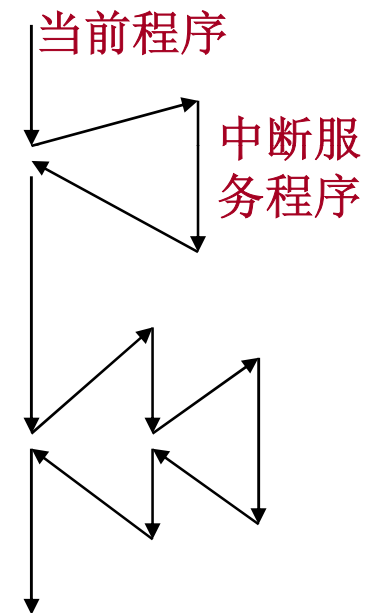


CPU与CP0的接口

- Special Instructions
 - mfc0
 - Move from coprocessor 0
 - mtc0
 - Move to coprocessor 0
 - eret
 - Return from exception (goes to EPC)
- CP0 Hazard: 两者不同步, 存在延迟槽
 - mfc0 k0, \$cause
 - nop /* mfc0指令执行速度慢, 延迟槽中加一个空操作 */
 - mov t0, k0 /* 将cause送t0, 进行下一步操作 */

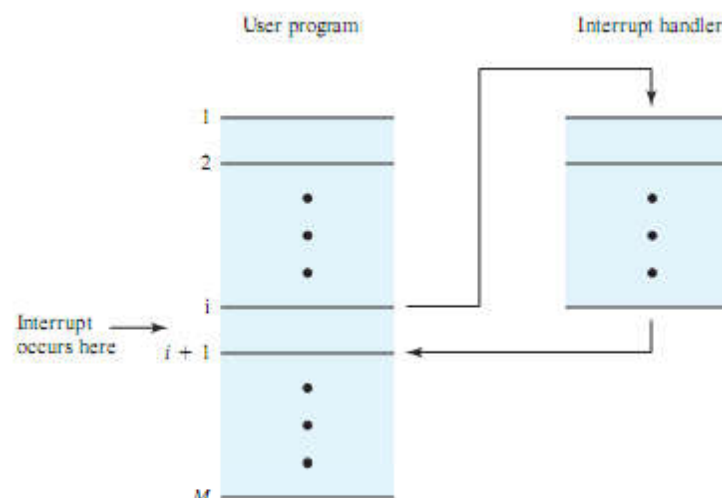
中断系统

- 中断的概念
 - 暂停当前程序的执行，转而执行其他程序，在它们执行完成后恢复被中断程序的执行。
 - 中断源：外部中断、内部中断（软中断、异常）
 - 中断源识别：查询法，向量法
 - 中断判优
 - 中断服务程序（ISR）：入口地址
 - 中断到达与响应时机
 - 中断响应
 - 断点（nPC）：系统关键状态，隐指令完成
 - 现场（regs, PSW）：中断服务相关
 - 中断返回（清中断）
 - 中断周期
 - 中断嵌套
- 中断机构
 - 为了实现中断，计算机系统中必须有相应的中断系统。



中断、异常、陷进、CALL

- 中断响应时机：程序的当前状态
 - 断点、现场？
 - 指令周期
 - 基于总线周期的计算机？
 - 冯式计算机：指令周期计算机
 - 流水线：实际上是总线周期计算机！
- 同步/异步：到达/发生时刻与指令周期的关系
 - 中断：异步，在中断周期响应；返回下一条指令
 - 何时不允许中断？
 - 保存断点，保存现场，恢复现场，中断返回；原子操作（临界区）
 - 异常：同步，无中断周期；返回当前指令或无法返回
 - 陷阱：同步，无中断周期；返回下一条指令
 - 过程调用：同步，无中断周期；返回下一条指令
- 中断/异常/陷阱改变系统状态，Call不改变系统状态



小结

- 中断周期要完成哪些微操作？
- 多周期状态机中，出现溢出的指令是否将错误结果写回？
- 多周期中状态机中，如何响应中断？
- 指令顺序执行，中断“精确”；指令流水执行，中断“精确”或“非精确”可选
- 流水线是否存在“中断周期”？
- 为何提交点是M段？
- EPC和cause应该在哪个段？异常检测电路？
- mips异常返回指令eret如何实现？
- 异常与中断同时发生，优先级？
- （分支）延迟槽中的指令发生异常，EPC = ？
- 比较中断、异常、陷阱、过程调用
 - 请求时间、响应时间，断点与现场，返回点，同步异步，中断周期、系统状态？
- 作业：唐（8.23、8.24）、4.25.1

