中国科学技术大学
University of Science and Technology of China

Ch 5 中断与异常

王 超

中国科学技术大学计算机学院
高能效智能计算实验室

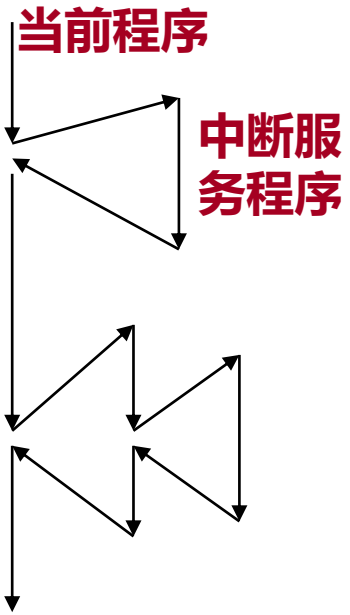
2026年春

内容提要

- 中断的基本概念（唐第五章、第八章）
- 中断要解决的若干问题（唐第五章、第八章）
- RISC-V流水线中断和异常的实现（COD）

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
  - ✓ I/O：响应外设的中断
    - 系统与环境交互：实时响应外部事件（I/O，人机交互）
  - ✓ 并发：提高计算机的整机效率
    - 单核多任务并发：允许单处理器“同时”执行多个任务
    - 多核多任务并行：允许多处理器交互（多处理器(核)间通信）
  - ✓ 服务：用户程序与OS间交互，Trap (陷阱)
    - 一种提供系统服务和保护的机制
- 中断管理
  - ✓ 中断服务程序ISR　Interrupt Service Routines

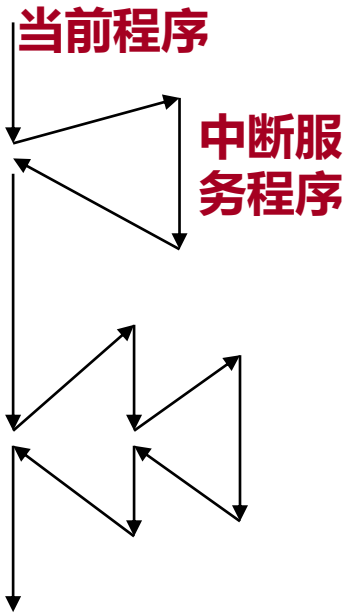


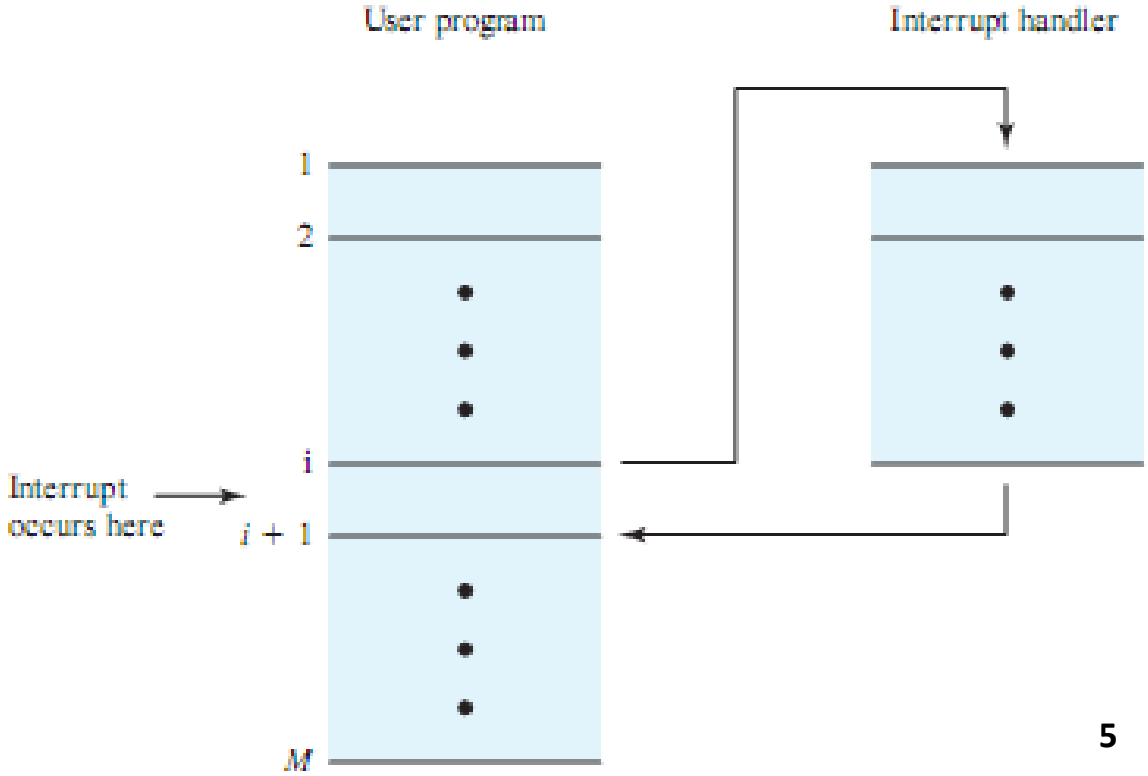
Exceptions、Interrupts、Traps

- x86：**Interrupt**，包含
  - ✓外部中断：硬中断（hardware interrupt）
    - 可屏蔽中断，不可屏蔽中断NMI(NonMaskable Interrupt)
  - ✓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**，包含
  - ✓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

中断的行为

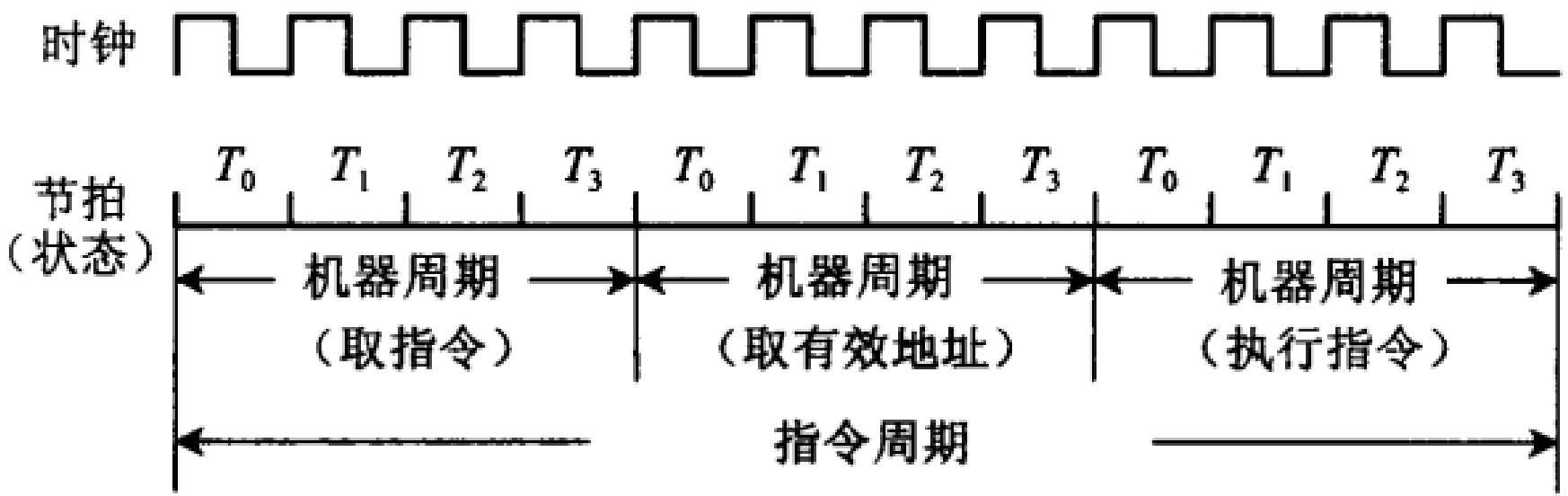
- 过程调用
- 对CPU控制的转移（当前程序=>中断服务程序）
- 对计算资源的并发使用





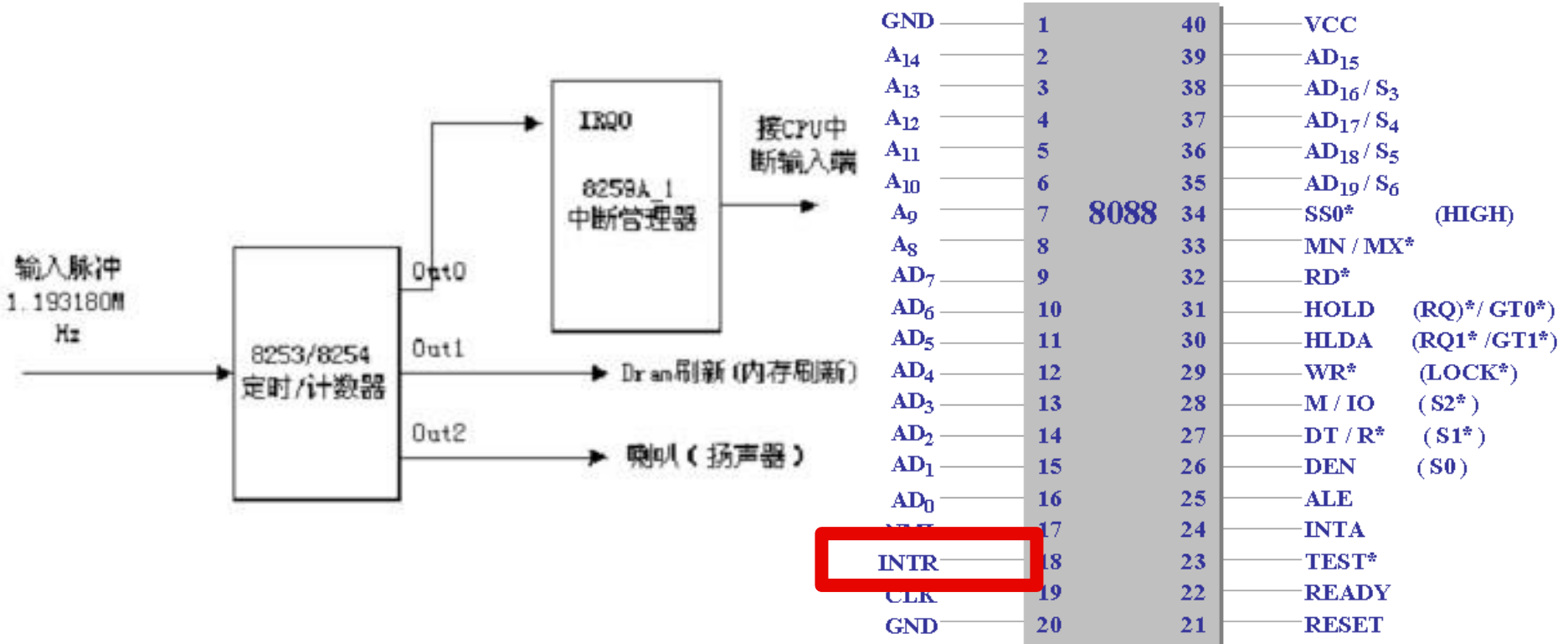
中断/异常发生的时机

- I/O中断 Interrupts（**随时发生，延迟响应**）
- 异常Exceptions（**随时发生，随时处理**）
- 陷阱Traps（**专用指令，特殊处理**）



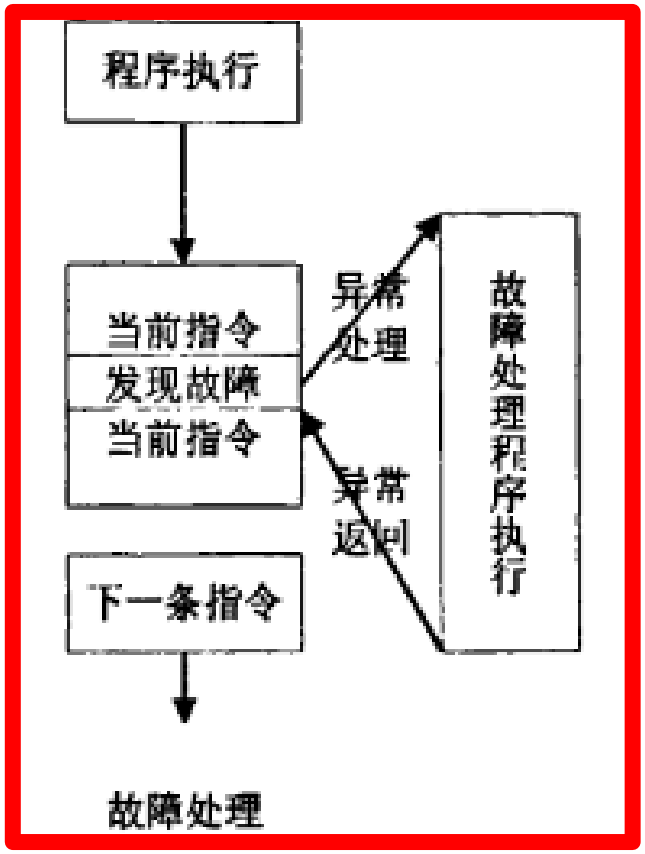
例：时钟中断

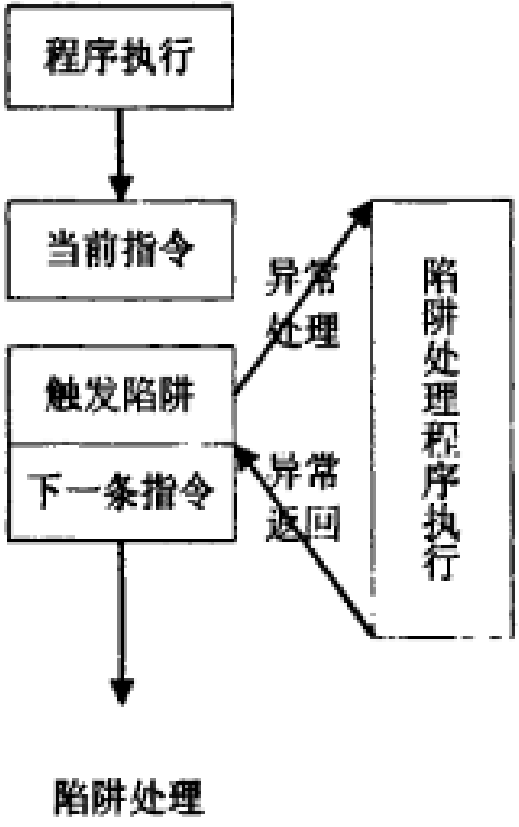
- 由可编程定时/计数器产生的INT
  - ✓维持系统时间(间隔大约10ms更新实时时钟--RTC)
  - ✓时钟中断维持系统时间、促使环境的切换，以保证所有进程共享CPU

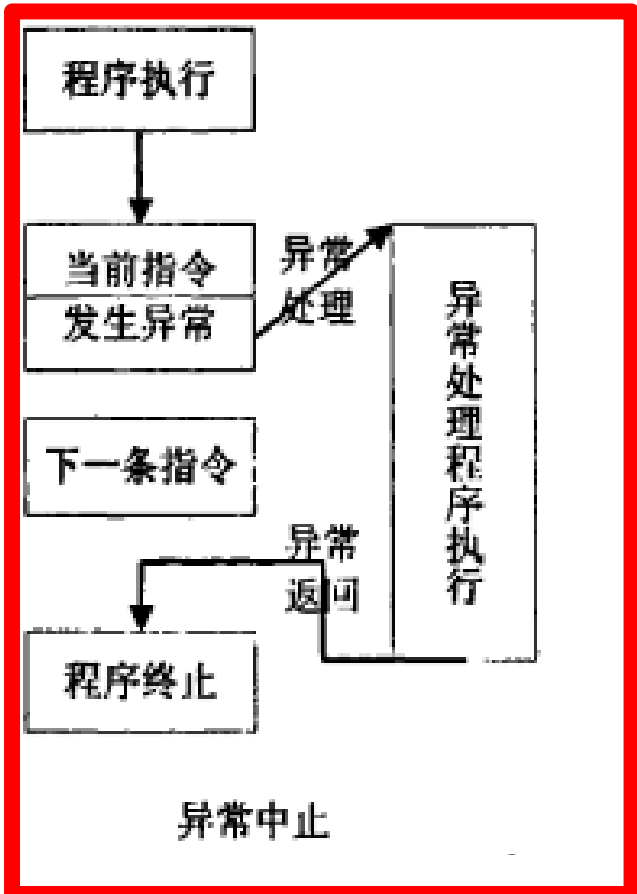


例:异常处理

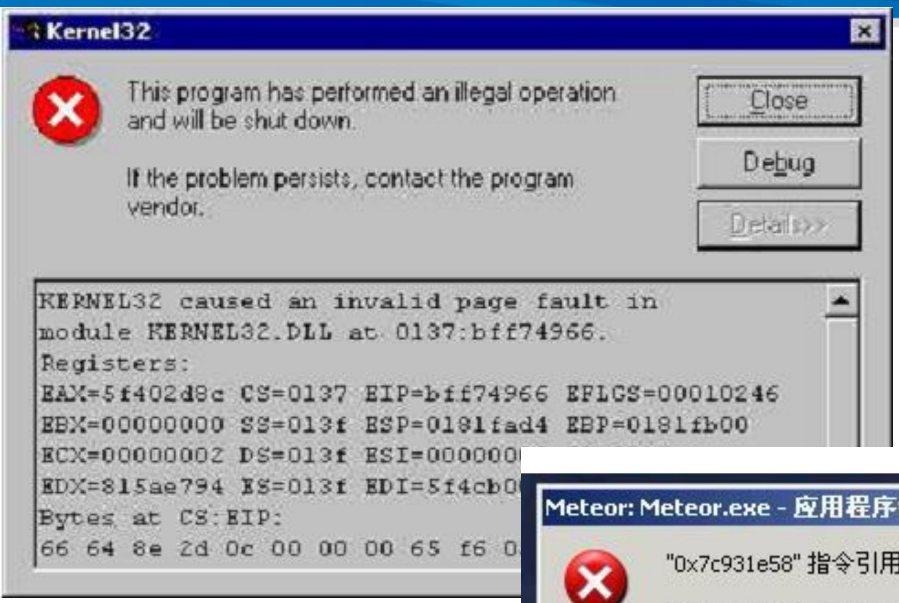
- 服务：将控制转移给OS的异常处理程序
  - ✓可恢复异常（下图中“故障”，OS处理后返回）
  - ✓不可恢复异常（异常中止）

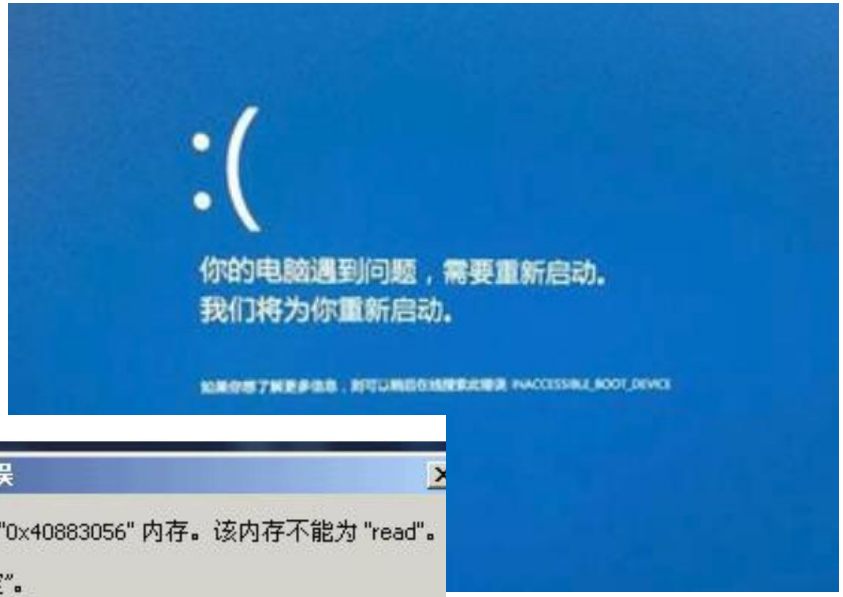


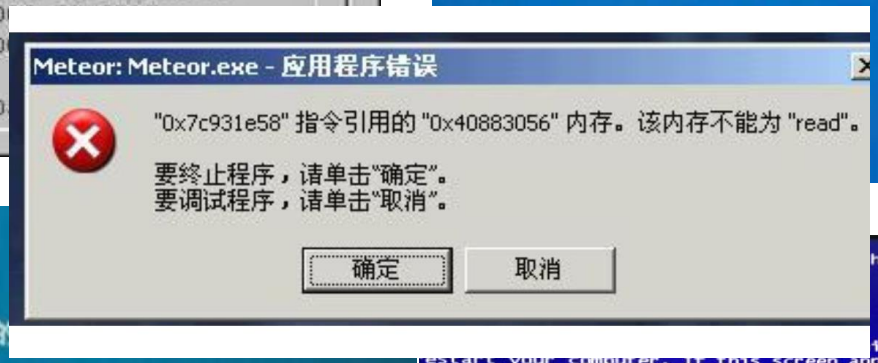


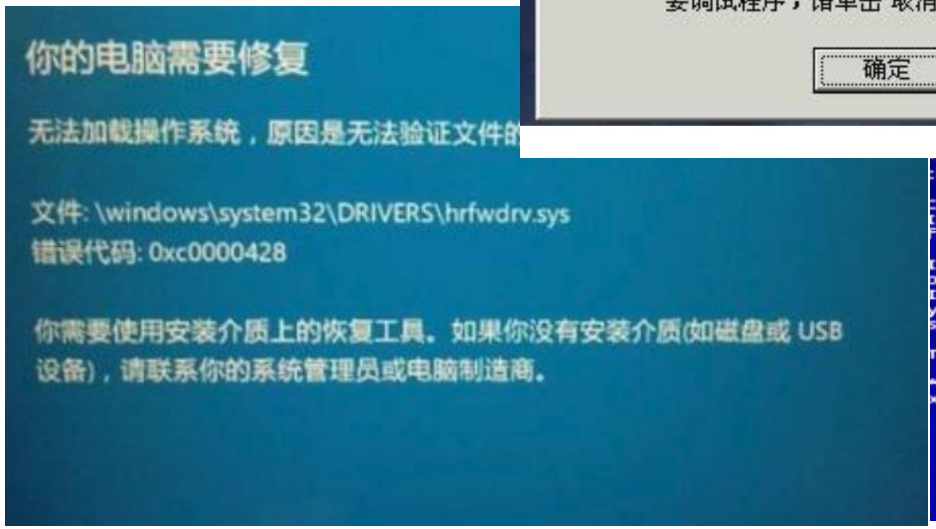


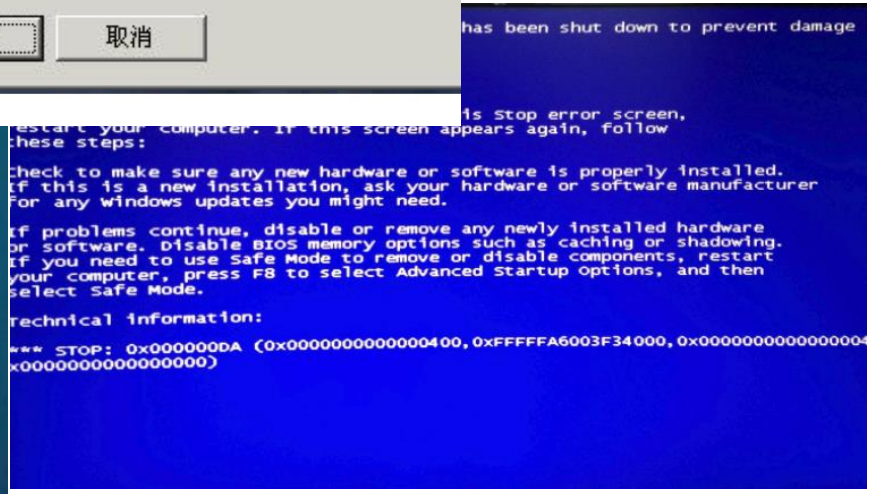
例：不可恢复异常：异常终止







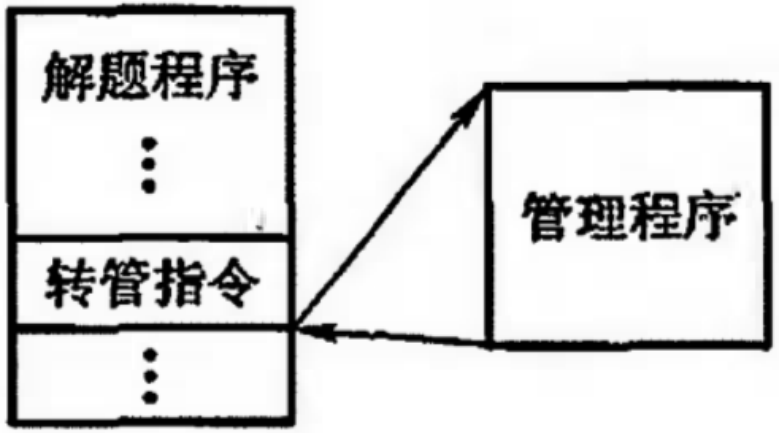




引起中断的因素

(1) 人为设置的中断

如转管指令（与内存、IO交互）



(2) 程序异常　　溢出、操作码不能识别、除法非法

(3) 硬件故障

(4) I/O 设备发起的中断请求

(5) 外部事件　　用键盘中断现行程序

中断系统需解决的问题

(1) 各中断源如何向CPU提出请求？

(2) 各中断源同时提出请求怎么办？

(3) CPU什么条件、什么时间、以什么方式响应中断？

(4) 如何保护现场？

(5) 如何寻找入口地址？

(6) 如何恢复现场，如何返回？

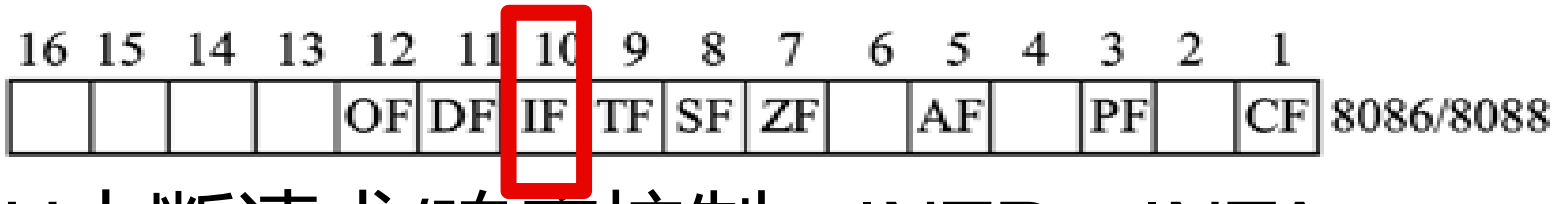
(7) 处理中断的过程中又出现新的中断怎么办？

解决方案：硬件 + 软件

0、中断机构组成

1. CPU中断禁止/允许：IF@PSW

PSW即程序状态字（程序状态寄存器），Program Status Word。



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

1. 中断请求标记

(1) 各中断源 如何 向 CPU 提出请求？

1. 中断请求标记 INTR

一个请求源对应一个INTR中断请求标记触发器

多个INTR组成中断请求标记寄存器

①INTR分散在各个中断源的接口电路中

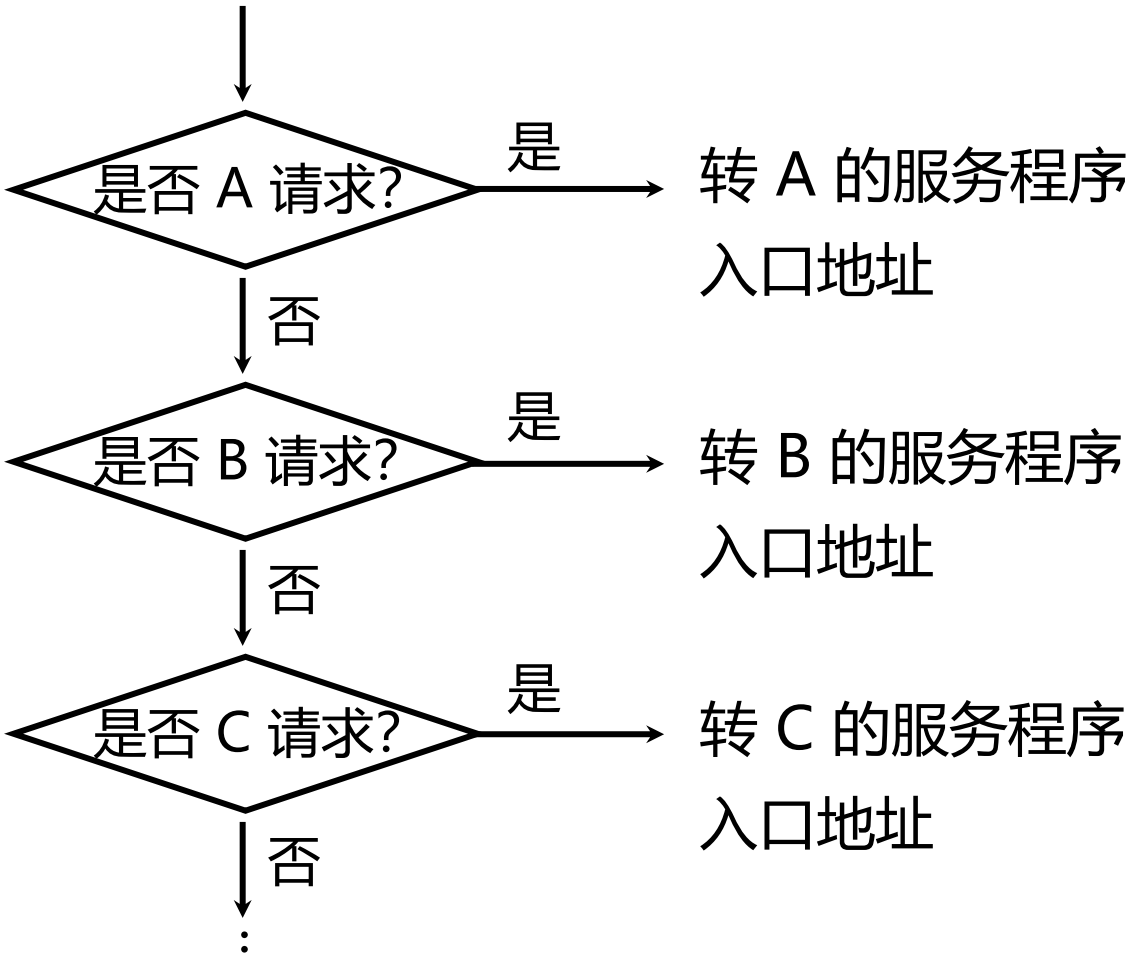
②INTR集中在CPU的中断系统内

(2) 各中断源 同时 提出 请求 怎么办？

2. 中断判优逻辑

(1) 软件实现（程序查询）

A、B、C 优先级按 降序 排列



2. 中断判优逻辑

(2) 硬件实现（排队器）

① 分散在各个中断源的接口电路中的链式排队器

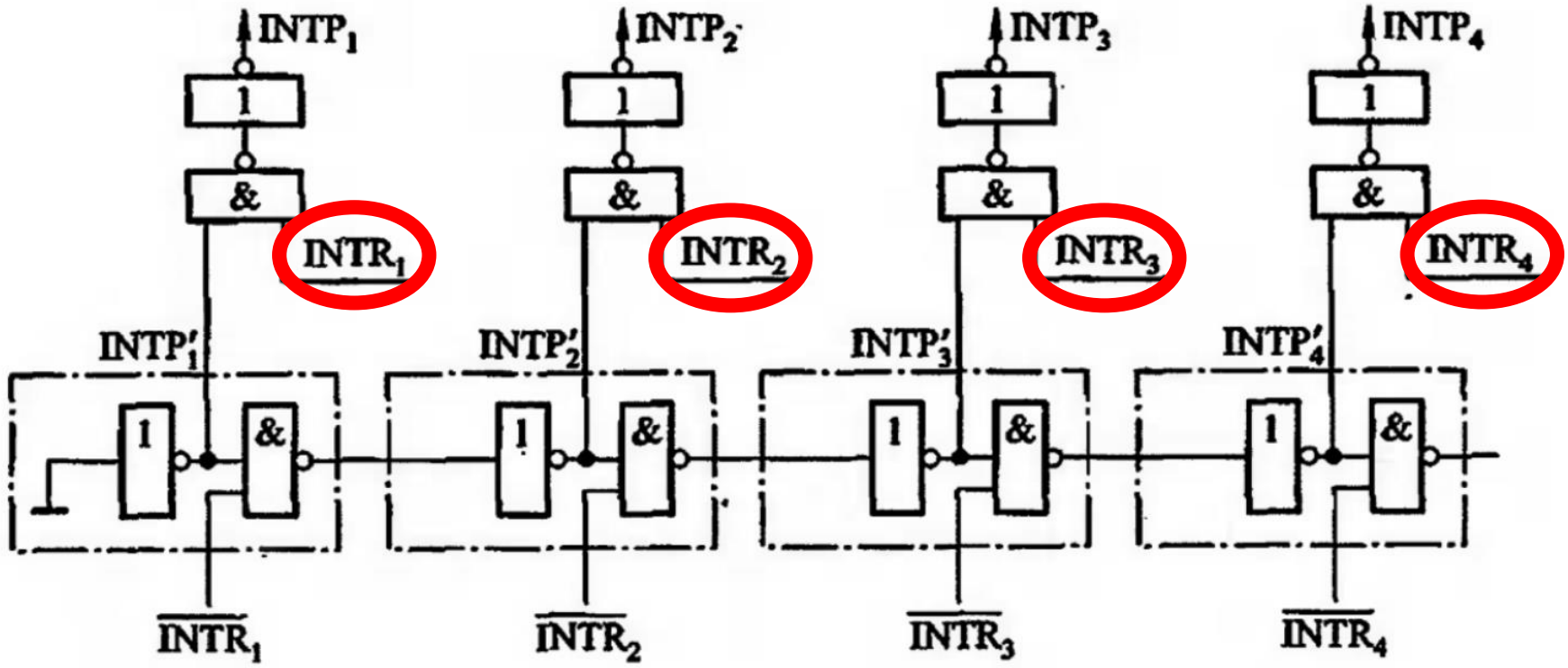
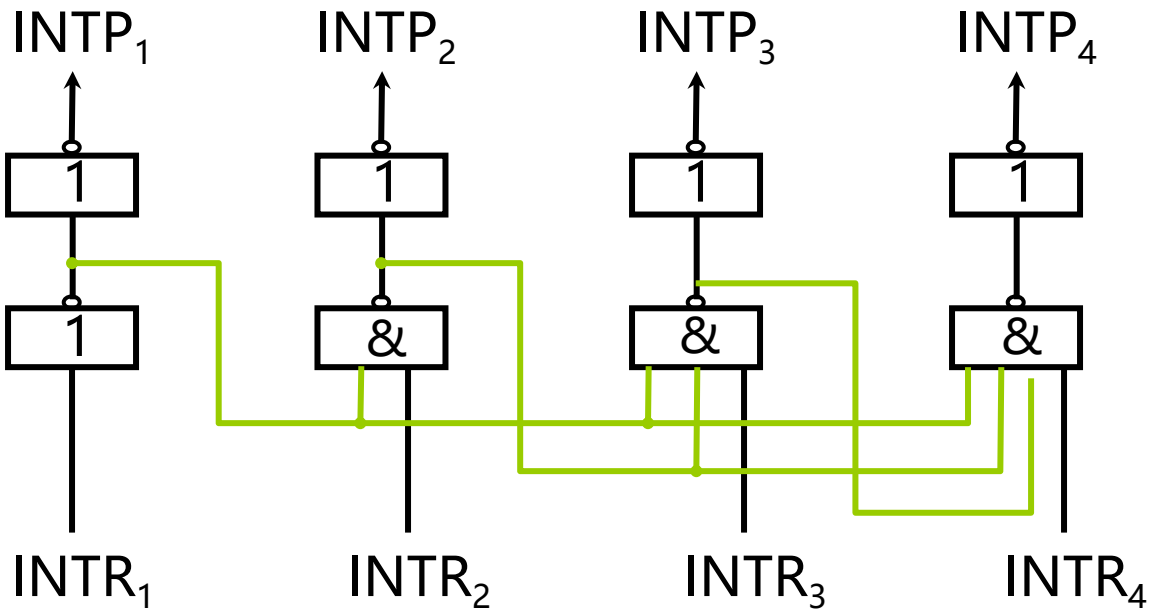


图 5.38　链式排队器

2. 中断判优逻辑

(1) 硬件实现（排队器）

② 集中在CPU内的链式排队器



$INTR_1$、$INTR_2$、$INTR_3$、$INTR_4$ 优先级按降序排列

中断响应

1. 响应中断的条件

CPU允许中断触发器　EINT = 1

2. 响应中断的时间

指令执行周期结束时刻由CPU 发查询信号

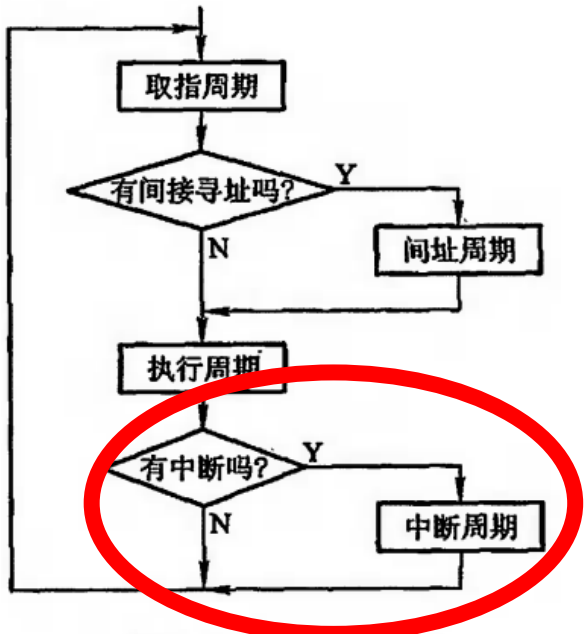


图 8.8　指令周期流程

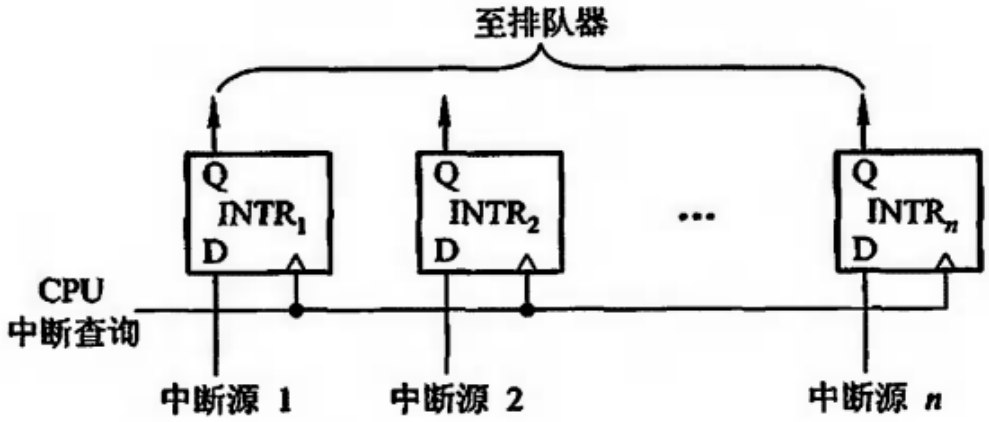


图 8.29　CPU 在统一时间发中断查询信号

3. 中断响应-中断隐指令

- 中断周期完成的主要操作，通过中断隐指令完成
- 1、保护程序断点
  - ✓断点存于特定地址（如0号地址）内，或者在堆栈
- 2、寻找服务程序入口地址
  - ✓向量地址->PC（硬件向量法）
  - ✓中断识别程序入口地址M->PC（软件查询法）
- 3、硬件关中断
  - ✓EINT 允许中断触发器 ➔ 0
  - ✓INT　中断标记触发器 ➔ 1

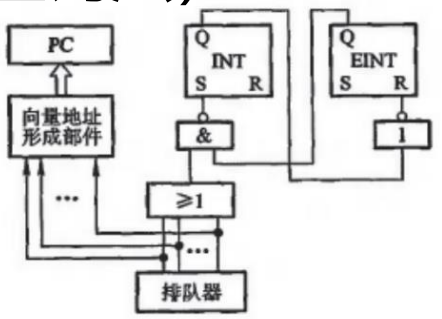
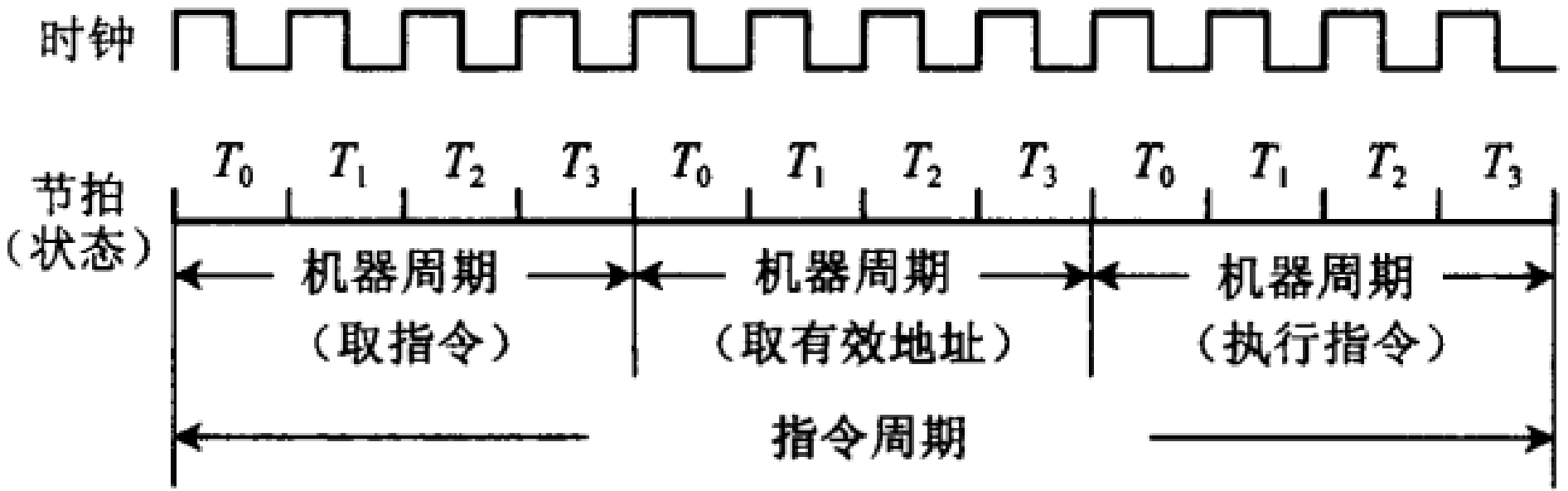


图 8.30　硬件关中断示意图

(3) CPU 什么条件、什么时间、以什么方式响应中断？

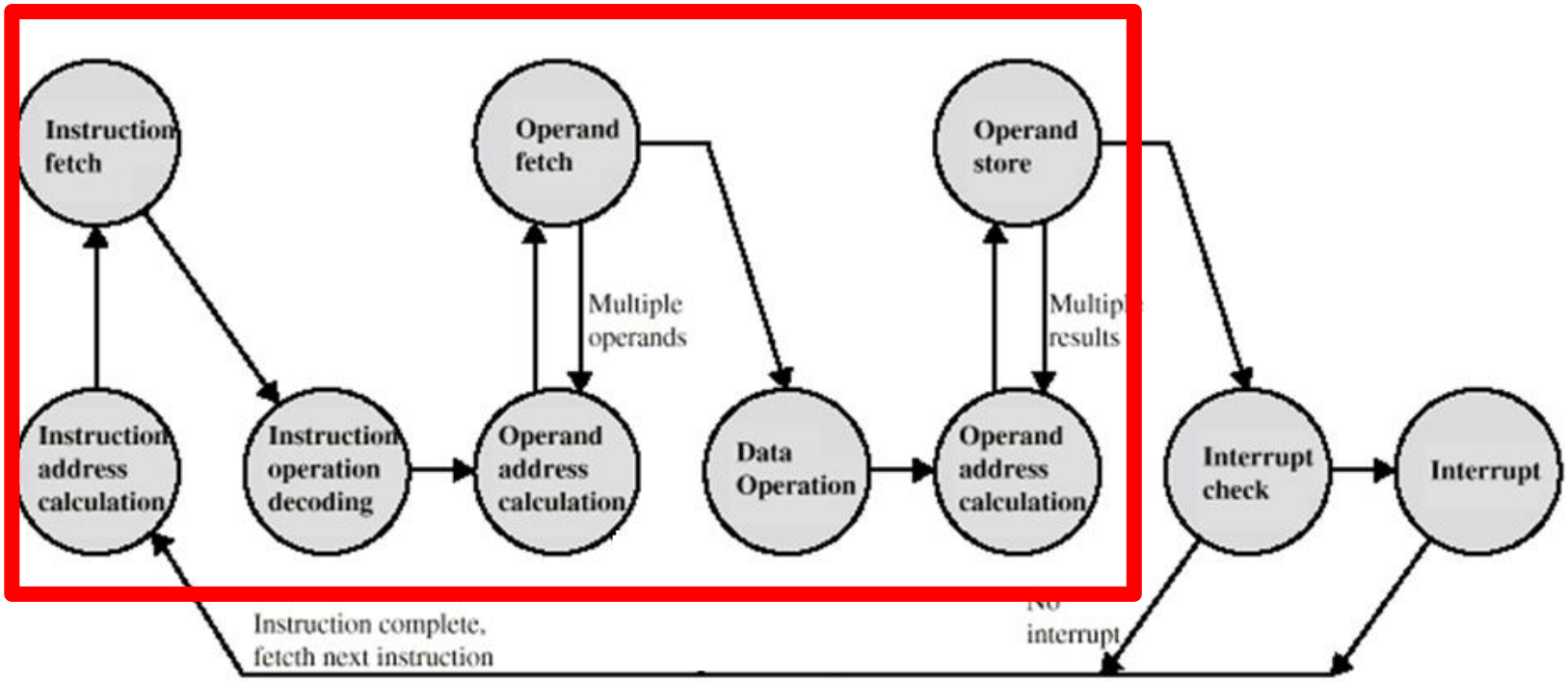
中断响应的时机与条件

- 中断发生与CPU响应时间
  - ✓外部中断（IO）：异步（延迟响应），指令周期结束
  - ✓内部中断（陷阱和异常）：同步（“马上”响应）



中断周期(Instruction Cycle with Interrupts)

- 指令周期结束
- CPU与中断控制器通信，响应外部事件(INTR,INTA)
  - ✓是否有中断请求
  - ✓识别中断源，获得中断向量
- 动作：存PC和PSW，关中断，转ISR



五、保护和恢复现场

(4) 如何 保护现场？

(6) 如何 恢复现场，如何 返回？

1. 保护现场
   - 断点 ——→ 中断隐指令完成
   - 寄存器 内容 ——→ 中断服务程序ISR完成

2. 恢复现场

中断服务程序

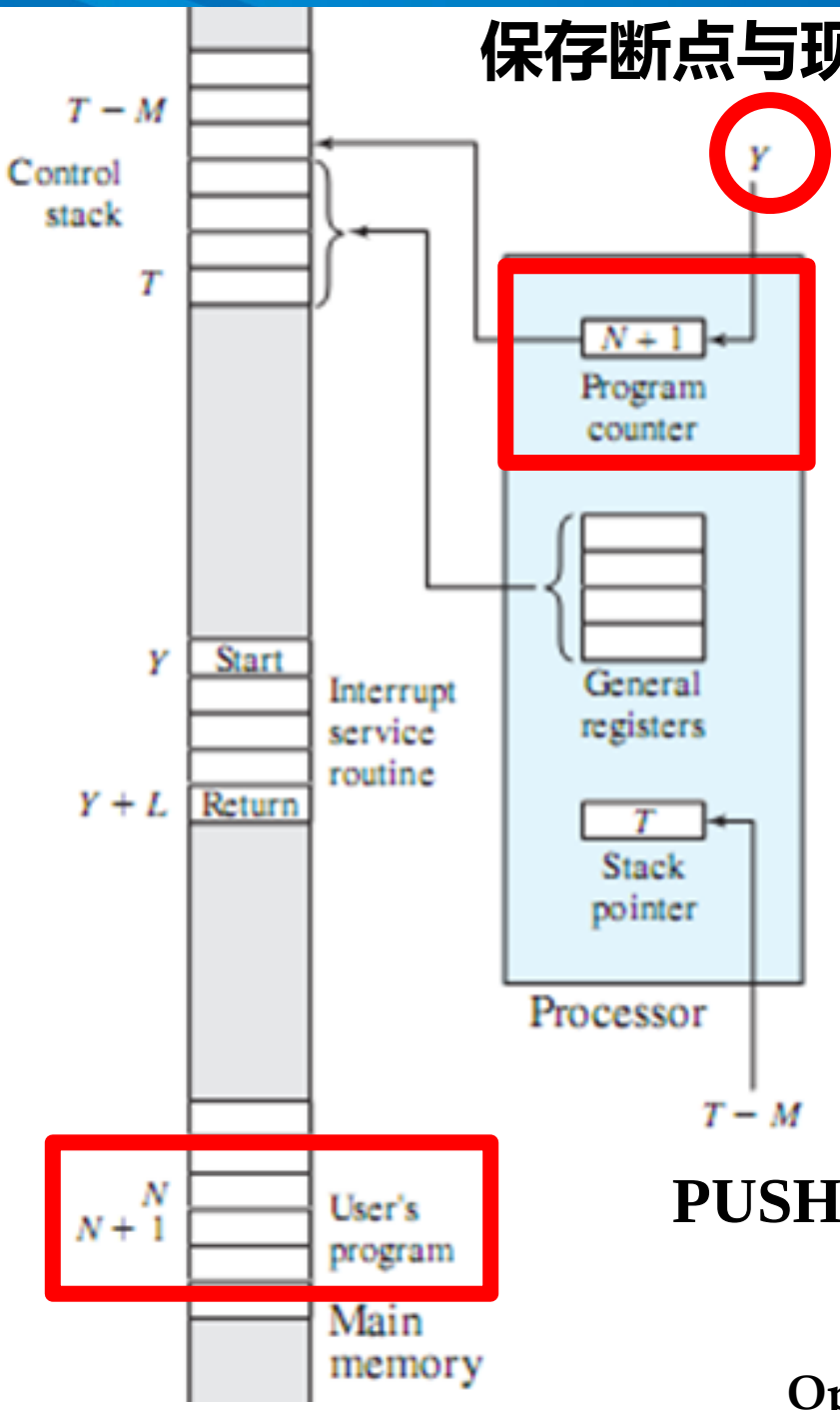
保护现场	PUSH
其它服务程序	视不同请求源而定
恢复现场	POP
中断返回	IRET

```
show proc near
 push ds
    push es
    push ax
    push cx
    push dx
    push bx
   push sp
    push bp
    push si
    push di
    sti
   mov dl,41h
    mov ah,02h
    int 21h
 exit :
   cli
   pop di
    pop si
    pop bp
    pop sp
    pop bx
    pop dx
    pop cx
    pop ax
    pop es
    pop ds
      iret
show endp
```

中断服务程序ISR示例

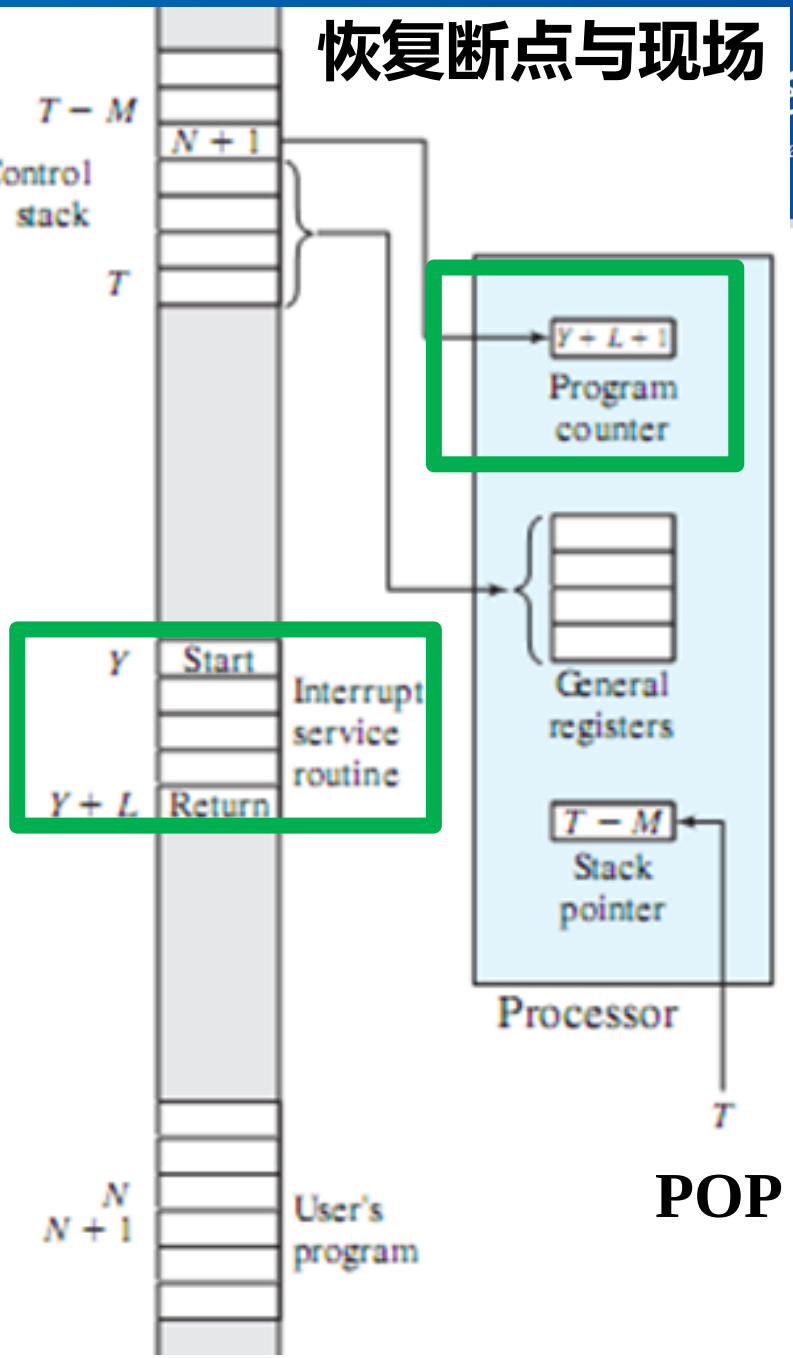
汇编用法参见 IBM PC的BIOS及DOS功能调用、微机原理及接口技术

保存断点与现场



PUSH

恢复断点与现场

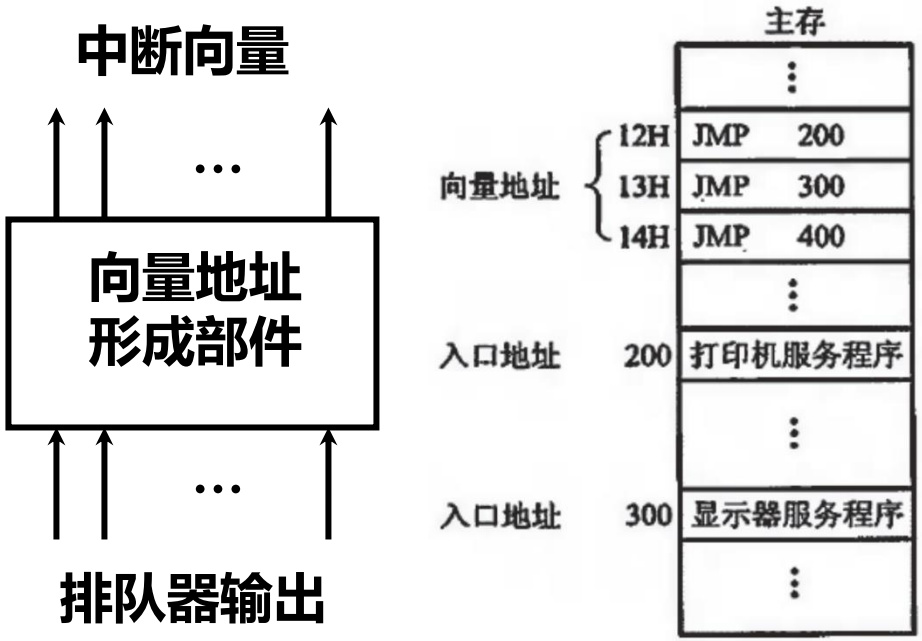


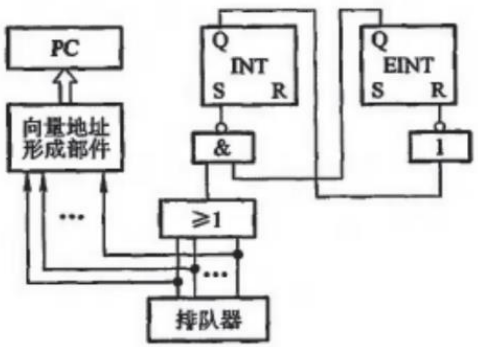
POP

(5) 如何 寻找入口地址？

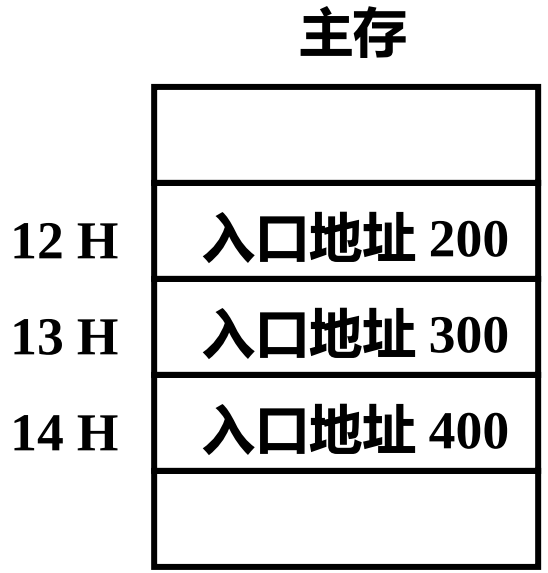
中断服务程序入口地址的寻找

1. 硬件向量法





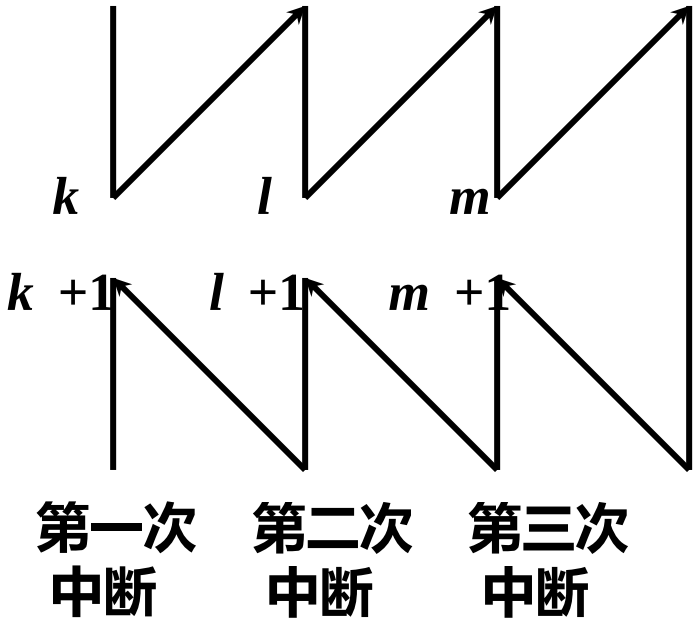
2. 软件查询法



向量地址　12H、13H、14H

入口地址　200、 300、 400

1. 多重中断的概念(中断嵌套)



程序断点 $k+1$， $l+1$， $m+1$

(7) 处理中断的过程中又 出现新的中断 怎么办？

2. 实现多重中断的条件

(1) 提前设置开中断

中断隐指令响应，存断点之后关中断
中断服务**结束之前**开中断

中断隐指令响应，存断点之后关中断
中断服务**开始之后**开中断

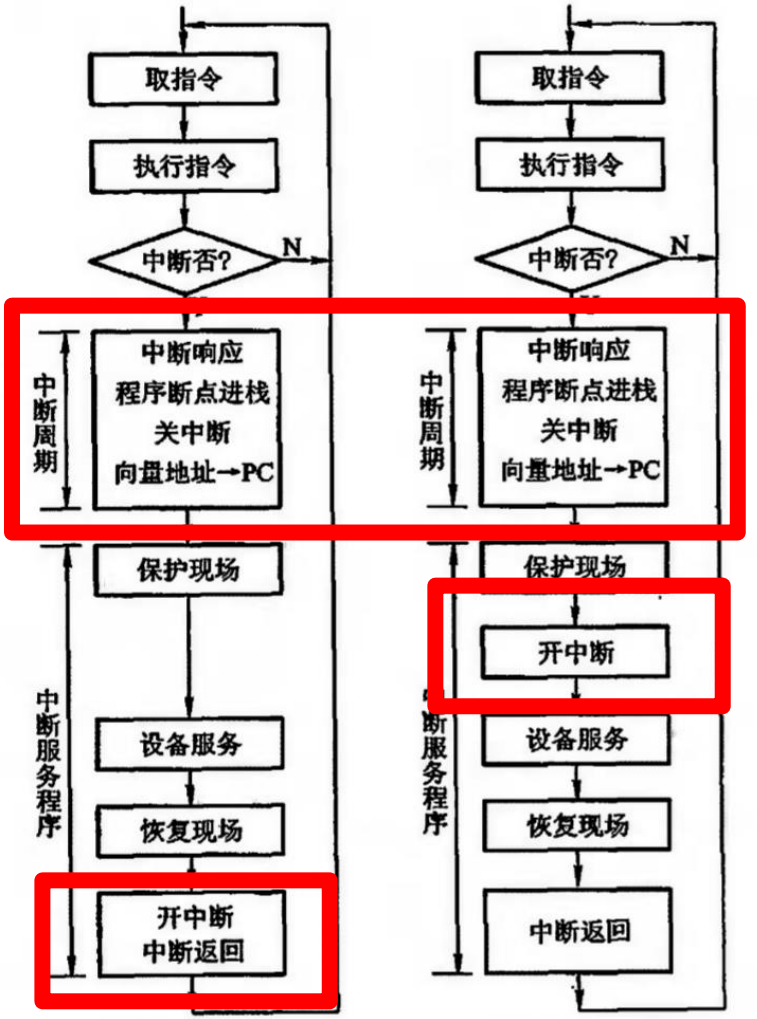


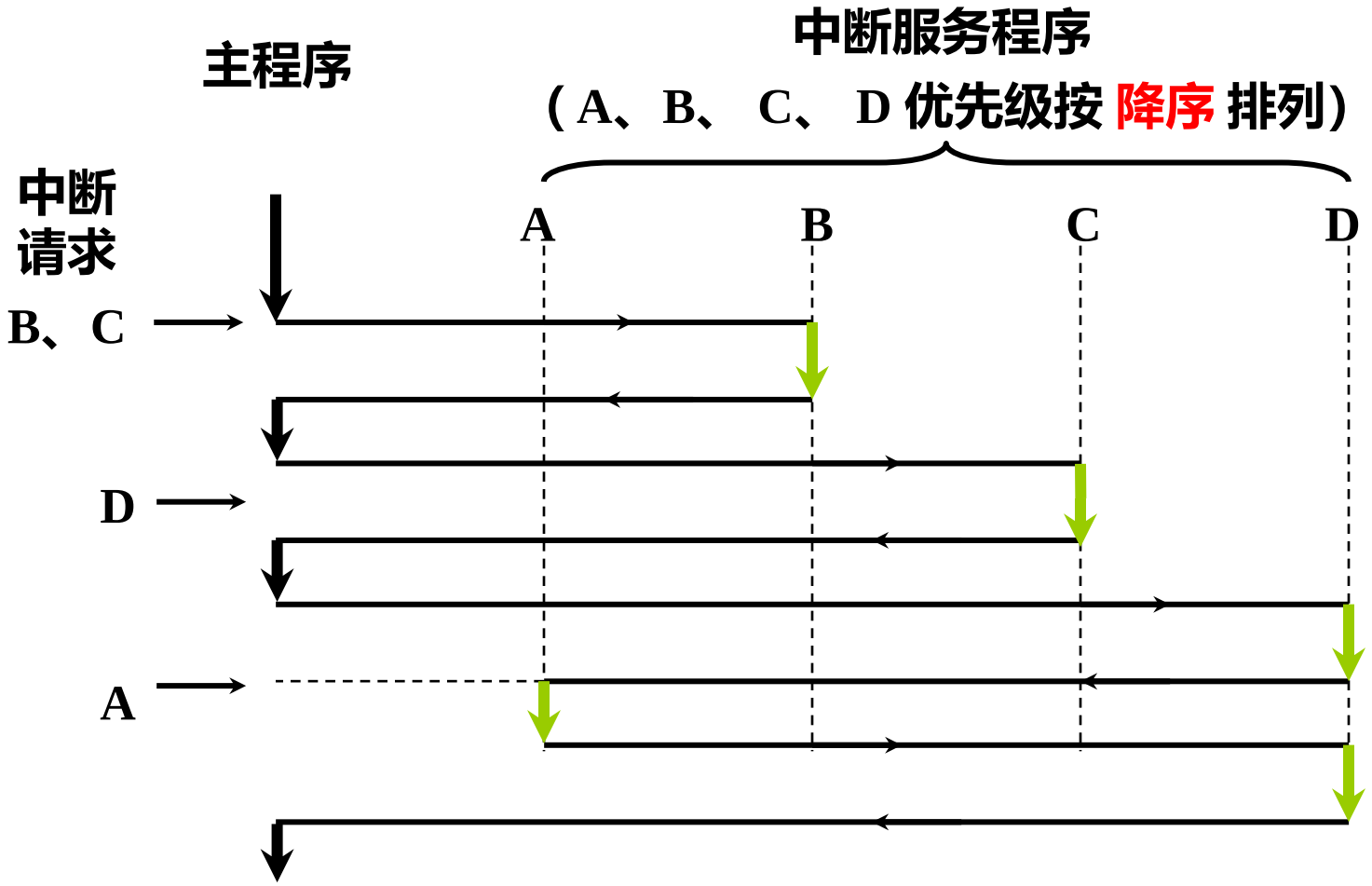
图 5.43 单重中断和多重中断服务程序流程

2. 实现多重中断的条件

(1) 提前设置开中断

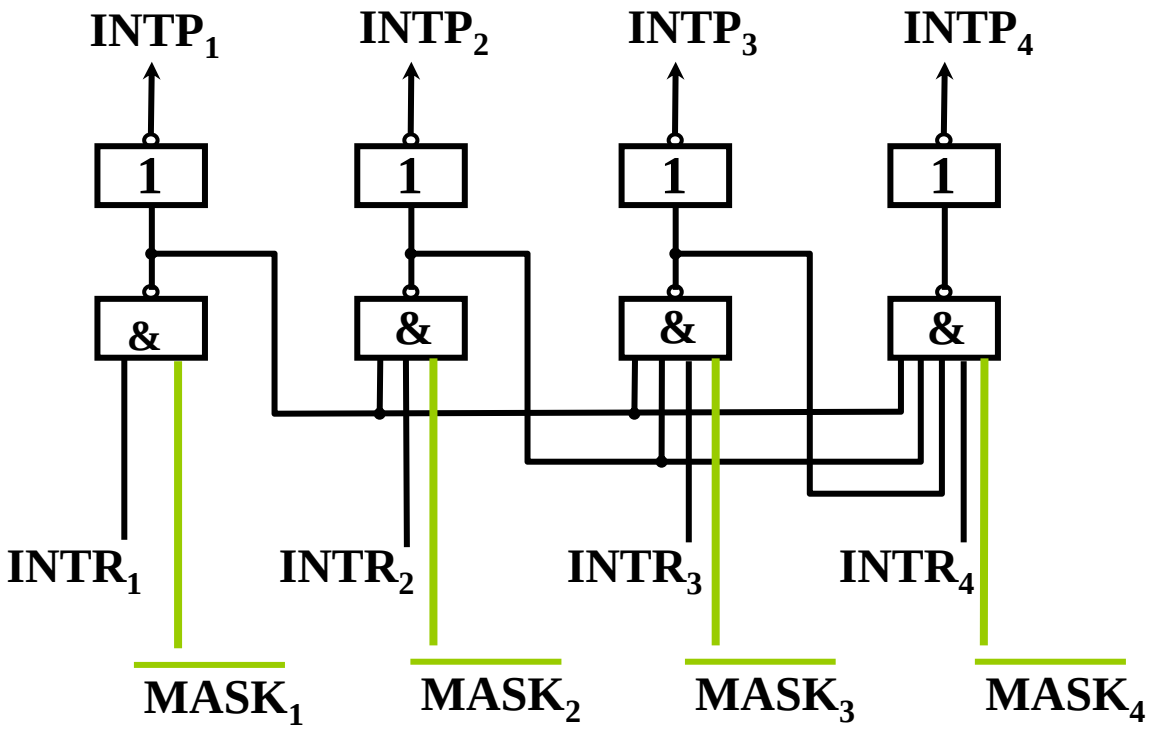
优先级如何设定?

(2) 优先级别高 的中断源 有权中断优先级别低 的中断源



3. 屏蔽技术

(1) 屏蔽触发器的作用-动态优先级设定



MASK = 0（未被屏蔽）　　$MASK_i$ = 1 （被屏蔽）

INTR 被置 “1”　　$INTP_i$ = 0 （不能被排队选中）

(2) 屏蔽字

16个中断源 1，2，3 … 16 按 降序 排列，每个对应16位屏蔽字

在ISR中设置屏蔽字，屏蔽对应中断源

优先级	屏 蔽 字
1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2	0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
3	0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
4	0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
5	0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
6	0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
⋮	⋮
15	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
16	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

(3) 新屏蔽字的设置-ISR

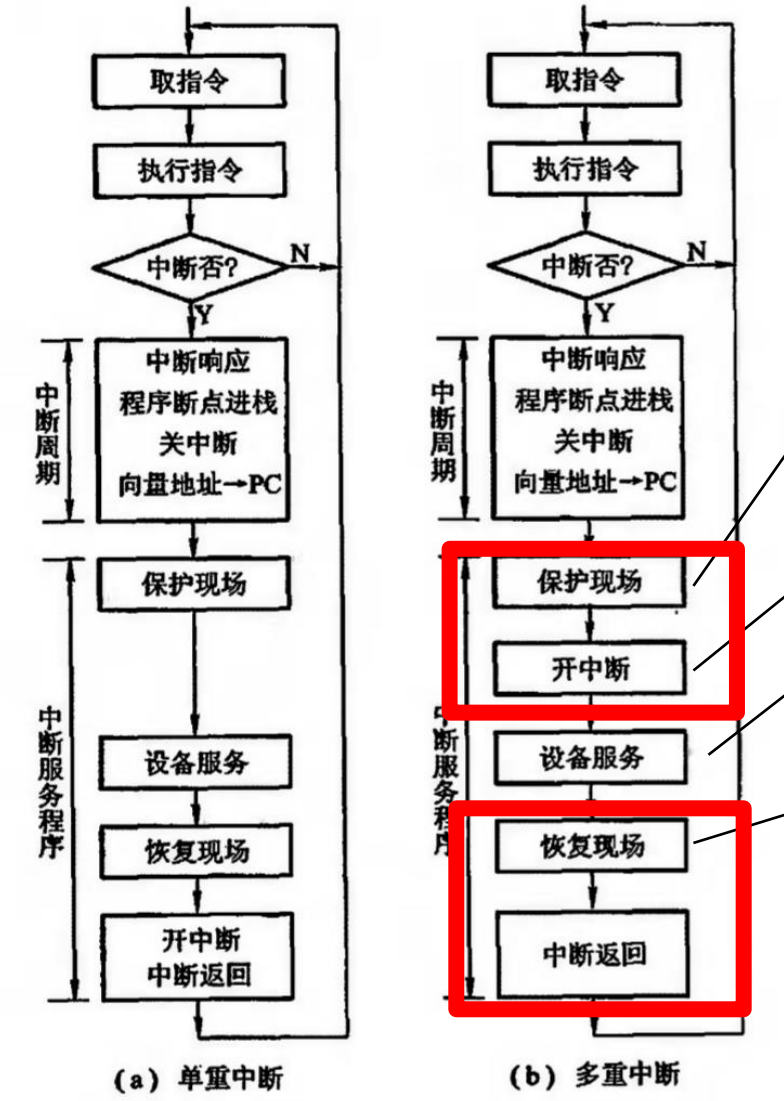
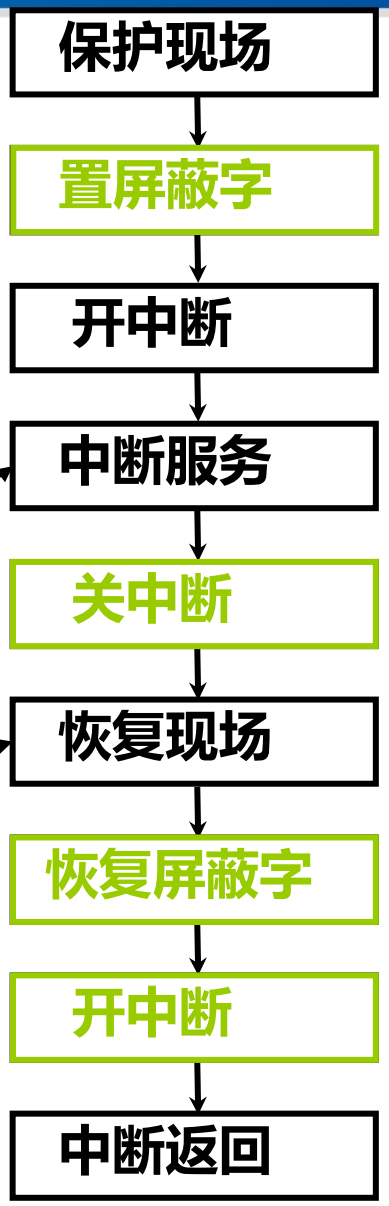


图 5.43 单重中断和多重中断服务程序流程



(4) 屏蔽技术可改变处理优先等级

响应优先级　　不可改变　为何不改变响应优先级？

处理优先级　　可改变（通过重新设置屏蔽字）

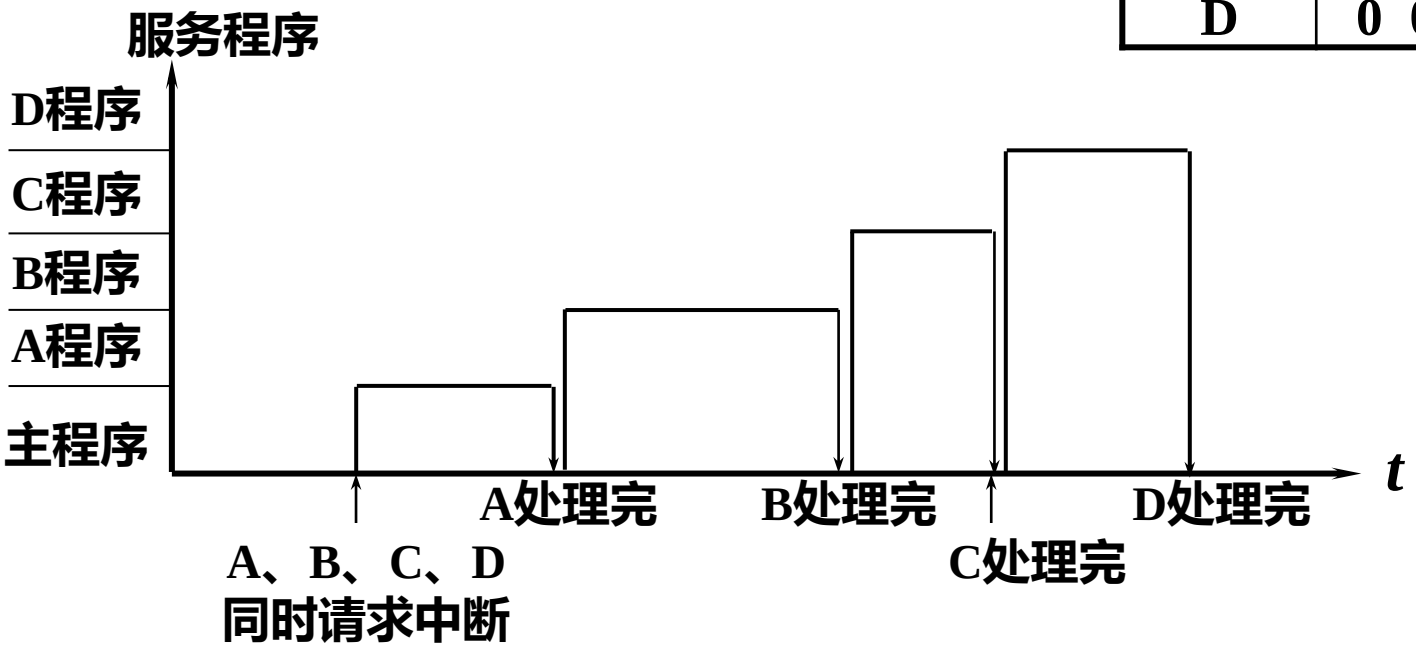
中断源	原屏蔽字	新屏蔽字
A	1 1 1 1	1 1 1 1
B	0 1 1 1	0 1 0 0
C	0 0 1 1	0 1 1 0
D	0 0 0 1	0 1 1 1

响应优先级　A→B→C→D 降序排列

处理优先级　A→D→C→B 降序排列

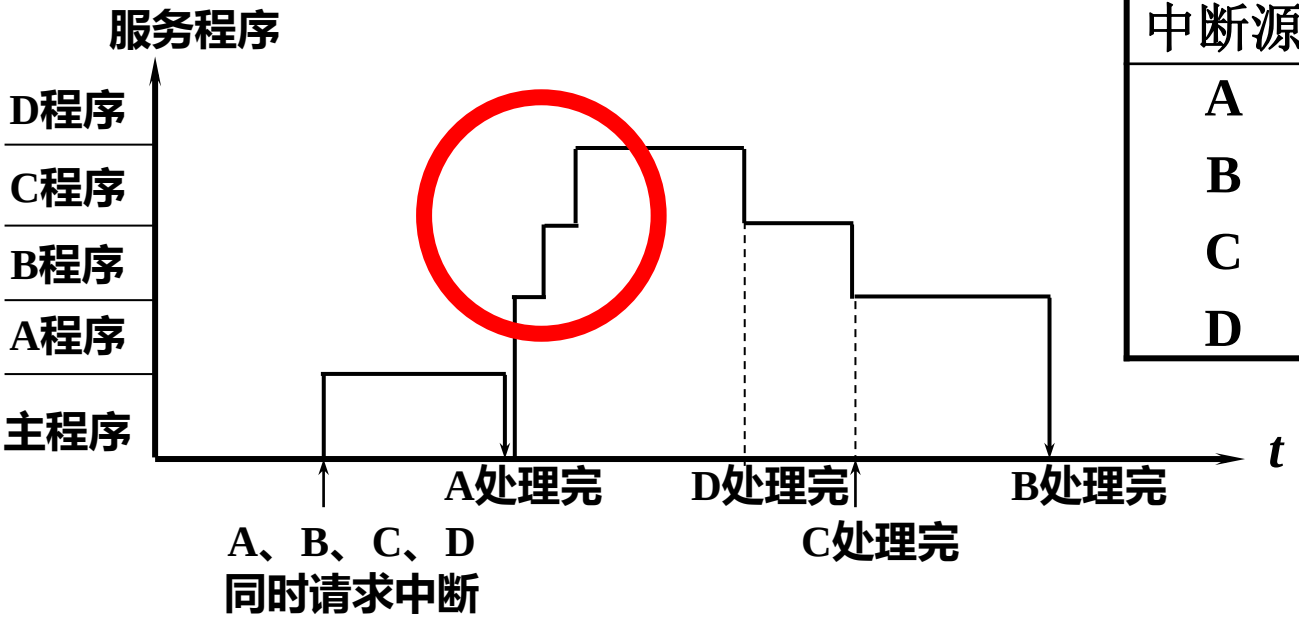
(4) 屏蔽技术可改变处理优先等级

中断源	原屏蔽字	新屏蔽字
A	1 1 1 1	1 1 1 1
B	0 1 1 1	0 1 0 0
C	0 0 1 1	0 1 1 0
D	0 0 0 1	0 1 1 1



CPU 执行程序轨迹（原屏蔽字）

(4) 屏蔽技术可改变处理优先等级



中断源	原屏蔽字	新屏蔽字
A	1 1 1 1	1 1 1 1
B	0 1 1 1	0 1 0 0
C	0 0 1 1	0 1 1 0
D	0 0 0 1	0 1 1 1

CPU 执行程序轨迹（新屏蔽字）

(4) 屏蔽技术的其他作用

可以 **人为地屏蔽** 某个中断源的请求，便于程序控制

问：结合屏蔽字设置的响应和处理流程，分析红色圈中的“锯齿”是从何来的?

总结：中断系统已解决的问题

(1) 各中断源如何向CPU提出请求？

(2) 各中断源同时提出请求怎么办？

(3) CPU什么条件、什么时间、以什么方式响应中断？

(4) 如何保护现场？

(5) 如何寻找入口地址？

(6) 如何恢复现场，如何返回？

(7) 处理中断的过程中又出现新的中断怎么办？

MIPS处理异常的方式

MIPS中的异常

0: Interrupt，中断；
1: TLB Modified，试图修改TLB中映射为只读的内存地址；
2: TLB Miss Load，试图读取一个没有在TLB中映射到物理地址的虚拟地址；
3: TLB Miss Store，试图向一个没有在TLB中映射到物理地址的虚拟地址存入数据；
4: Address Error Load，试图从一个非对齐的地址读取信息；
5: Address Error Store，试图向一个非对齐的地址写入信息；
6: Instruction Bus Error，一般是指令Cache出错；
7: Data Bus Error，一般是数据Cache出错；
8: Syscall，由syscall指令产生。操作系统下，通用的由用户态进入内核态的方法。
9: Break Point，由break指令产生。最常见的bp指令，是由编译器产生的，在除法运算时插入一个break point指令，以达到在除0时抛出错误信息的目的。因此，如果在定位问题时发现了一个Break Point异常，且它的异常分代码为07，应当考虑是出现了除0的情形；
10: RI，保留指令。在CPU执行到一条没有定义的指令时，进入此异常；
11: Co-processor Unavailable，协处理器不可用。这个异常是由于试图对不存在的协处理器进行操作引起的。特别的，在没有浮点协处理器的处理器上执行这条命令，会导致这个异常。随之，操作系统会调用模拟浮点的lib库，来实现软件的浮点运算；
12: **Overflow**，算术溢出。只有带符号的运算会引起这个异常；
13: Trap，这个异常来源于trap指令。和syscall指令类似地，trap指令也会引起一个异常，但trap指令可以附带一些条件，这样可以用于调试程序用。
14: VCEI，指令高速缓存中的虚地址一致性错误。
15: Float Point Exception，浮点异常；
16: Co-processor 2 Exception，协处理器2的异常；
17~22，留作扩展；
23: Watch，内存断点异常。当设定了WatchLo/WatchHi两个寄存器时起作用。当load/store的虚拟地址和WatchLo/WatchHi中匹配时，会引发这样一个异常
24~30，留作扩展；

MIPS中的异常分类

- 外部事件
  - IO中断或读总线错等 **0,6,7**
- 操作系统调用
  - 缺页或越界等8,9,13
- 算数溢出 12
- 非法指令10
- 其他硬件错误
  - 存储等1,2,3,4,5,14,23
  - 协处理器 11,16
  - **浮点 15**

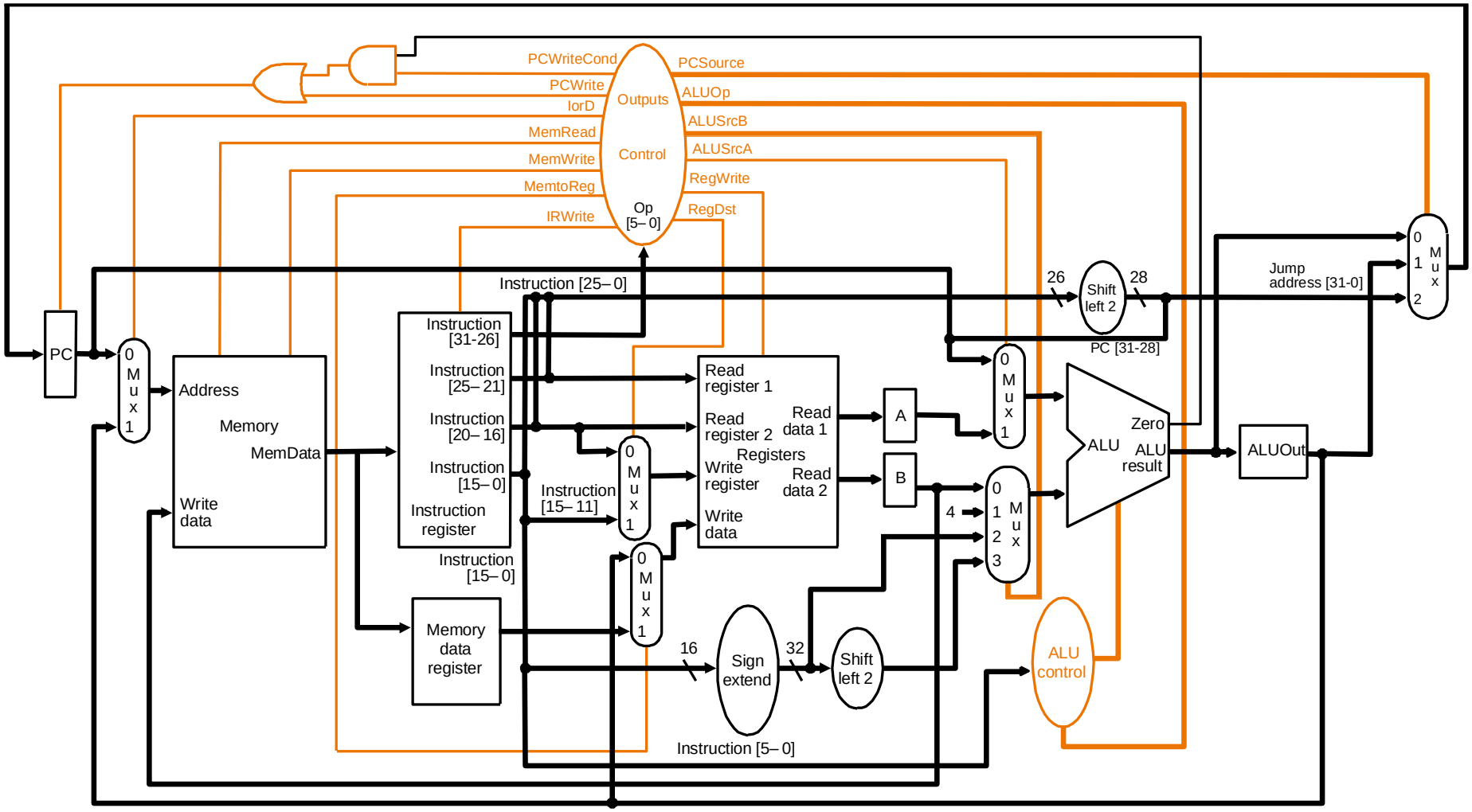
Type of event	From where?	MIPS terminology
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Arithmetic overflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

MIPS的异常处理

- 讨论两种异常作为示例：非法指令和算术溢出
- 异常处理的主要工作
  - ✓断点保存：将异常执行指令的地址保存在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：转到特定服务程序
    - 溢出：对溢出进行响应
  - ✓跳转异常服务程序（PC）：停止程序的执行并报告错误等
    - 未定义指令异常处理在$8000\ 0000_{hex}$
    - 算数溢出异常处理$8000\ 0180_{hex}$
- 服务程序的入口

Exception type	Exception vector address (in hex)
Undefined instruction	$8000\ 0000_{hex}$
Arithmetic overflow	$8000\ 0180_{hex}$

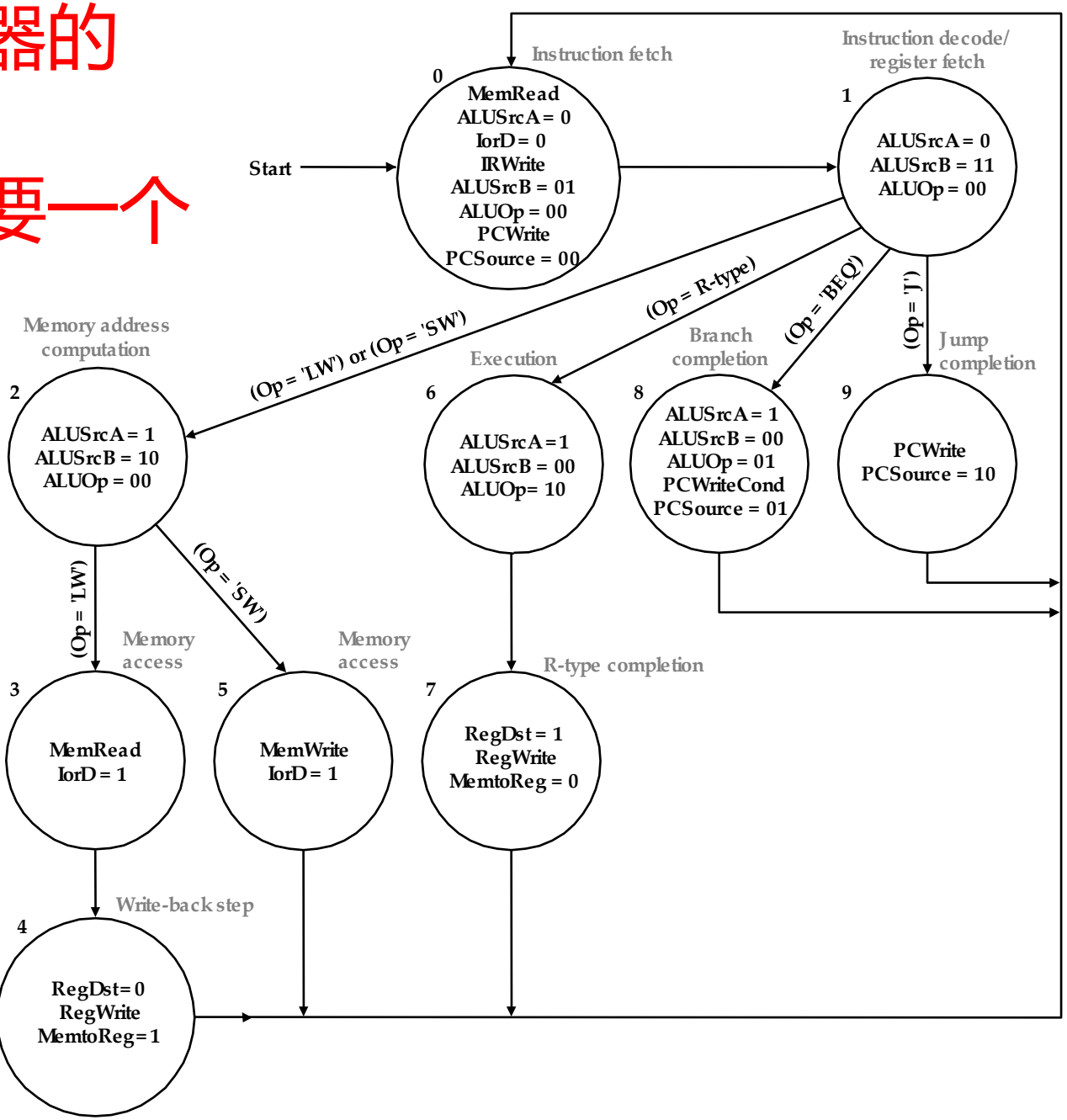
数据通路（单？多？流）


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5– 0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instruction [25– 0]
26
Shift
left 2
28
Jump
address [31-0]
PC [31-28]
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction
[31-26]
Instruction
[25– 21]
Instruction
[20– 16]
Instruction
[15– 0]
Instruction
register
Instruction
[15– 0]
Instruction
[15– 11]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
ALU
Zero
ALU
result
ALUOut
Memory
data
register
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5– 0]

多周期RTL

Step	R-Type	lw/sw	beq/bne	j
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
MEM	Reg[IR[15-11]] = ALUOut	MDR=Mem[ALUOut] Mem[ALUOut] = B		
WB		Reg[IR[20-16]] = MDR		

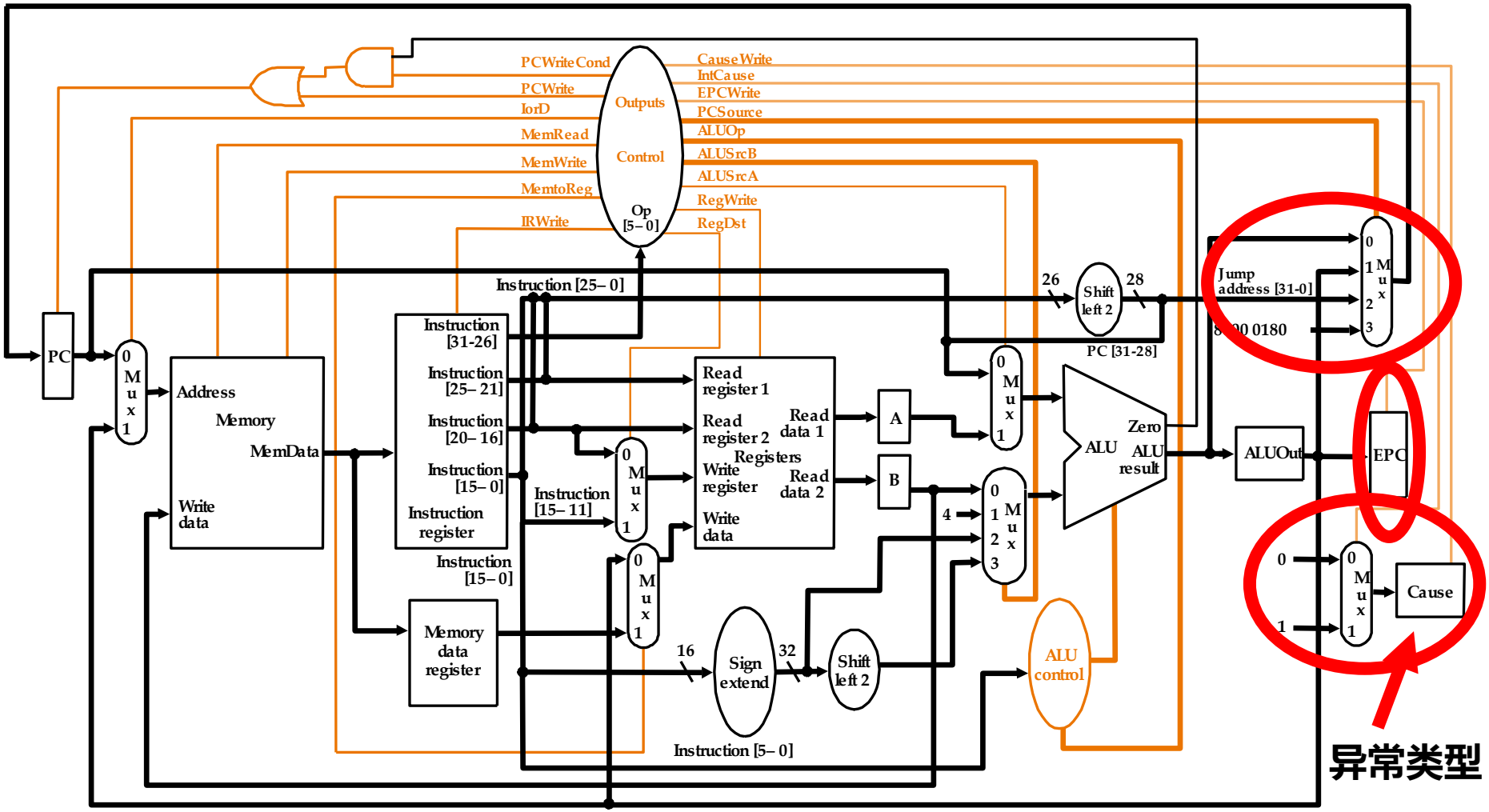
多周期控制器的MooreFSM，每个状态需要一个时钟周期。



复习：多周期支持异常

- 完成功能（非法指令、算术溢出）
  - ✓ PC赋值(根据异常原因跳转到异常处理程序)
  - ✓ 出错PC保存（PC-4保存到寄存器）
  - ✓ 出错原因保存（异常原因保存到寄存器）
- 数据通路
  - ✓ PC、EPC、Cause
- 控制信号
  - ✓ PC写（多路选择器）、EPC计算写入、Cause写入
- RTL代码
  - ✓ 非法指令实现、算术溢出实现
- 状态机
  - ✓ 加入几个状态，哪里加入？

多周期模式的异常处理


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op [5–0]
CauseWrite
IntCause
EPCWrite
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
PC
0 M u x 1
Address
Memory
MemData
Write data
Instruction [31-26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–0]
Instruction register
Instruction [15–0]
Instruction [15–11]
Instruction [25–0]
Memory data register
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
ALU
Zero
ALU result
ALUOut
EPC
Cause
26
Shift left 2
28
PC [31-28]
Jump address [31-0]
0180
16
Sign extend
32
Shift left 2
ALU control
Instruction [5–0]
异常类型

Exceptions

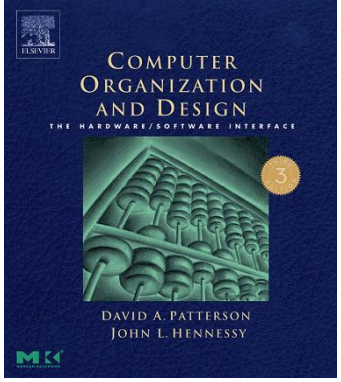
Step	R-Type	lw/sw	beq/bne	j	Exception
IF	IR = Mem[PC] PC = PC + 4				
ID	A = Reg[IR[25-21]] B = Reg[IR[20-16]] ALUOut = PC + (SE(IR[15-0]) << 2)				
EX	ALUOut = A op B	ALUOut = A +SE(IR[15-0])	If (A==B) then PC = ALUOut	PC = PC[31-28] || (IR[25-0]<<2	PC=0X80000000 EPC=PC-4 CAUSE=0
MEM	Reg[IR[15-11]] = ALUOut	MDR=Mem[ALUOut] Mem[ALUOut] = B			
WB		Reg[IR[20-16]] = MDR			

异常指令

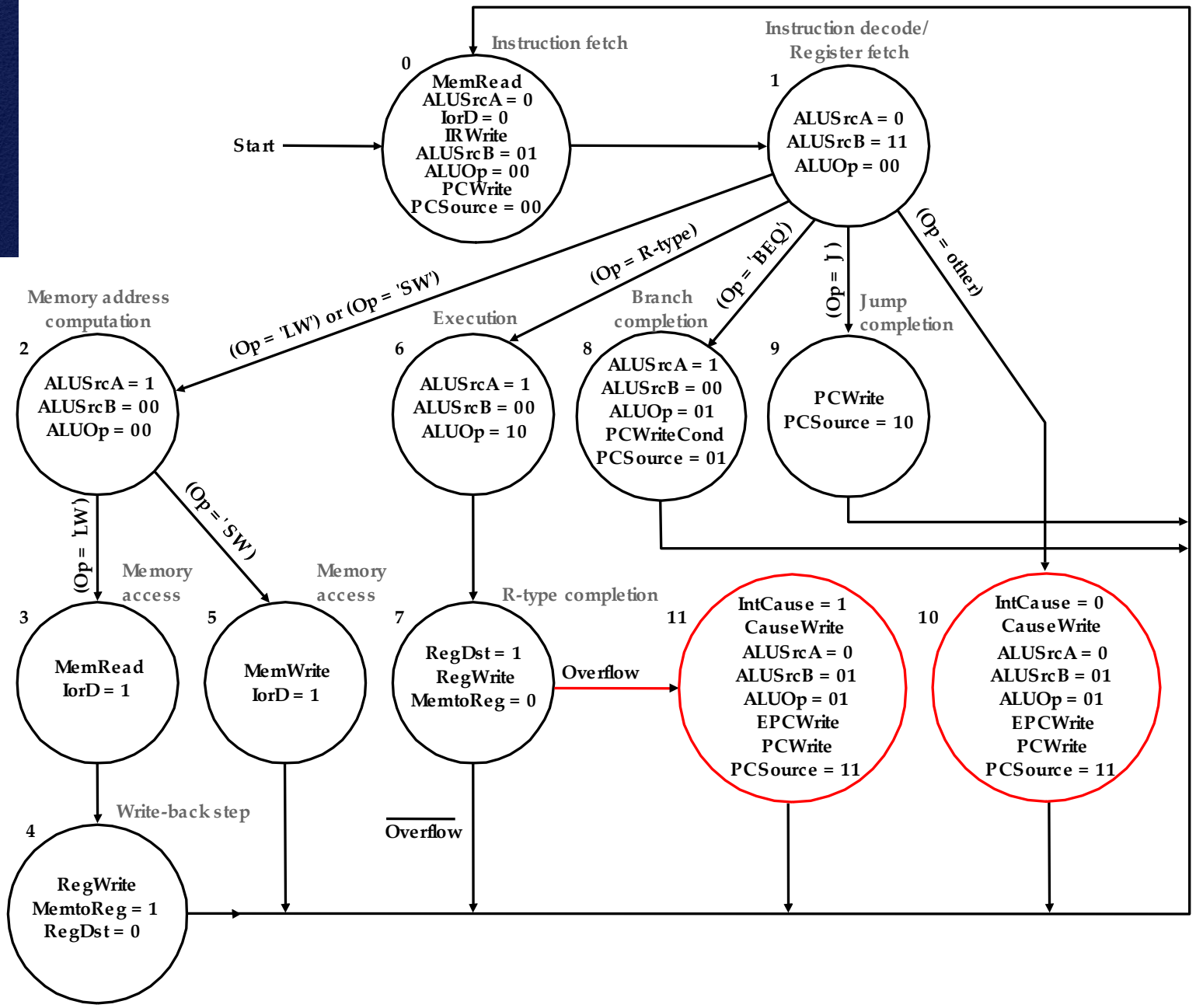
算术溢出
PC=0X80000180
EPC=PC-4
CAUSE=1

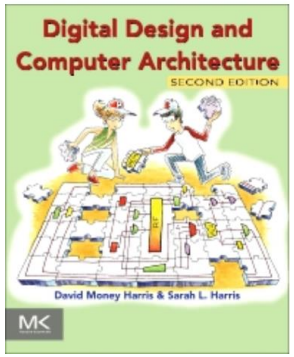
两种异常的处理

问：算术溢出应该在哪个周期?

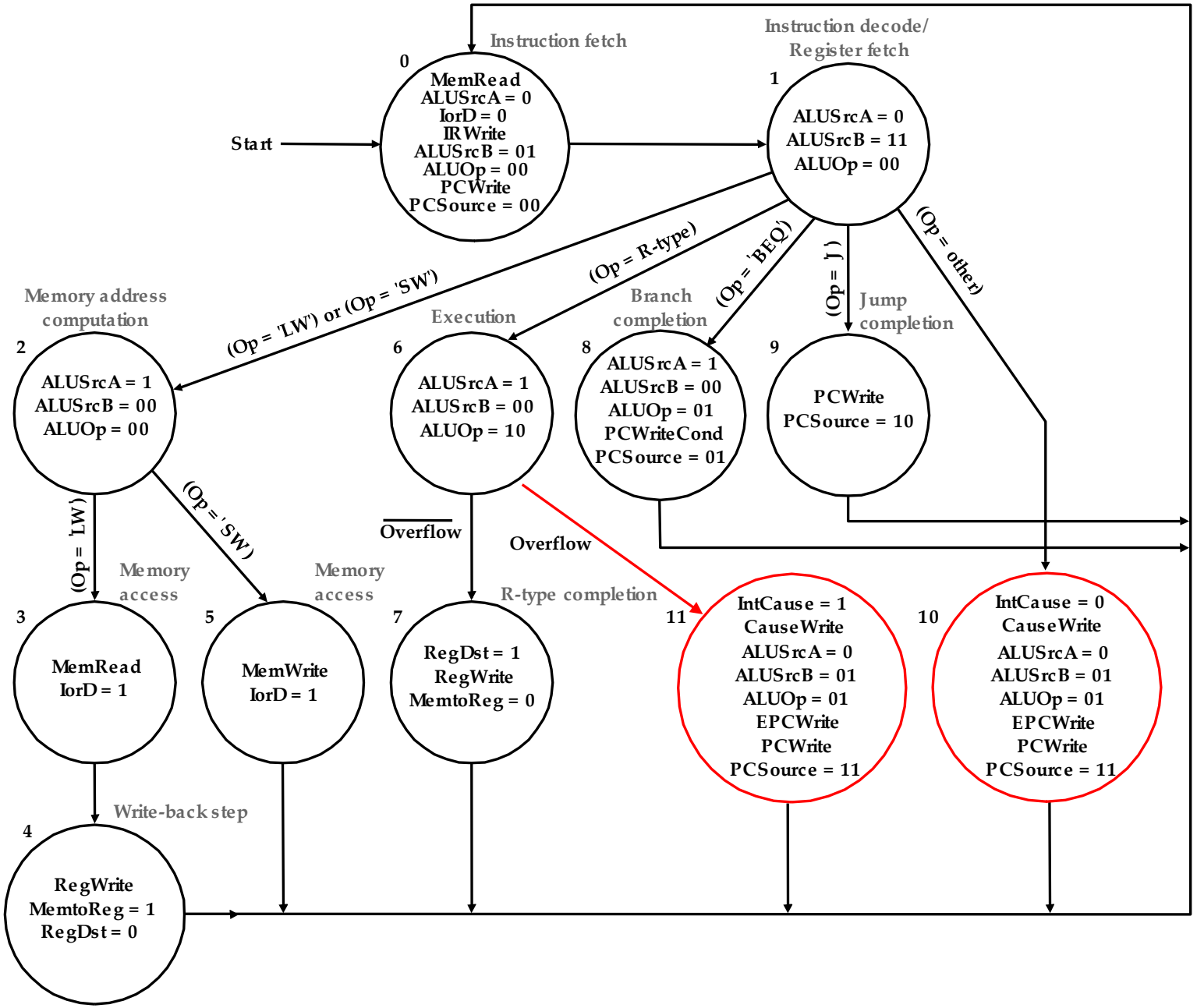

COMPUTER
ORGANIZATION
AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
DAVID A. PATTERSON
JOHN L. HENNESSY

异常处理控制


Start
Instruction fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Instruction decode/
Register fetch
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
(Op = other)
Memory address
computation
2
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 00
Execution
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
Branch
completion
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Jump
completion
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
3
MemRead
IorD = 1
Memory
access
5
MemWrite
IorD = 1
R-type completion
7
RegDst = 1
RegWrite
MemtoReg = 0
Overflow
11
IntCause = 1
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
10
IntCause = 0
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
Write-back step
4
RegWrite
MemtoReg = 1
RegDst = 0
Overflow

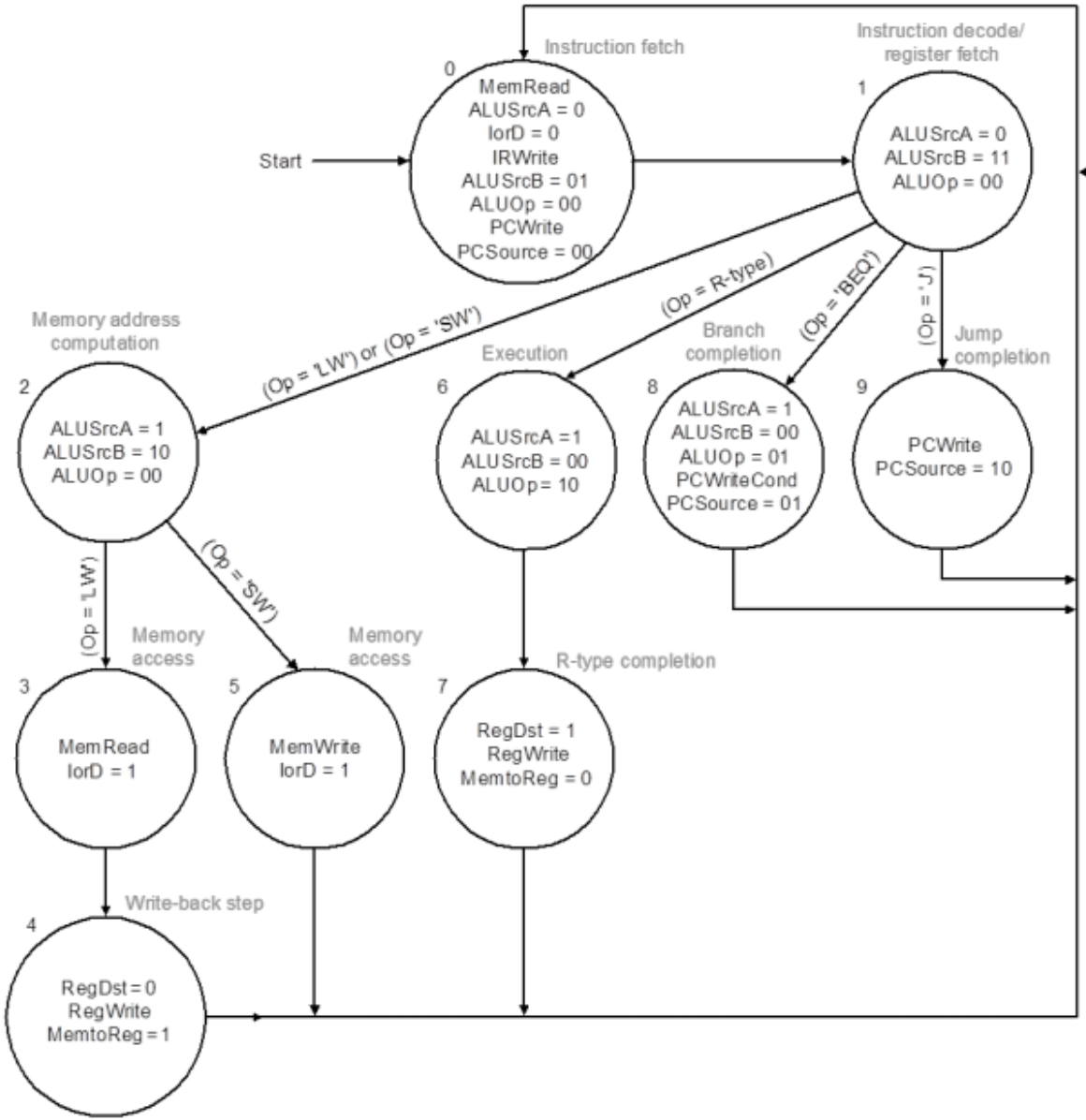


异常处理控制



Digital Design and Computer Architecture, Second Edition，David Harris， Sarah Harris.

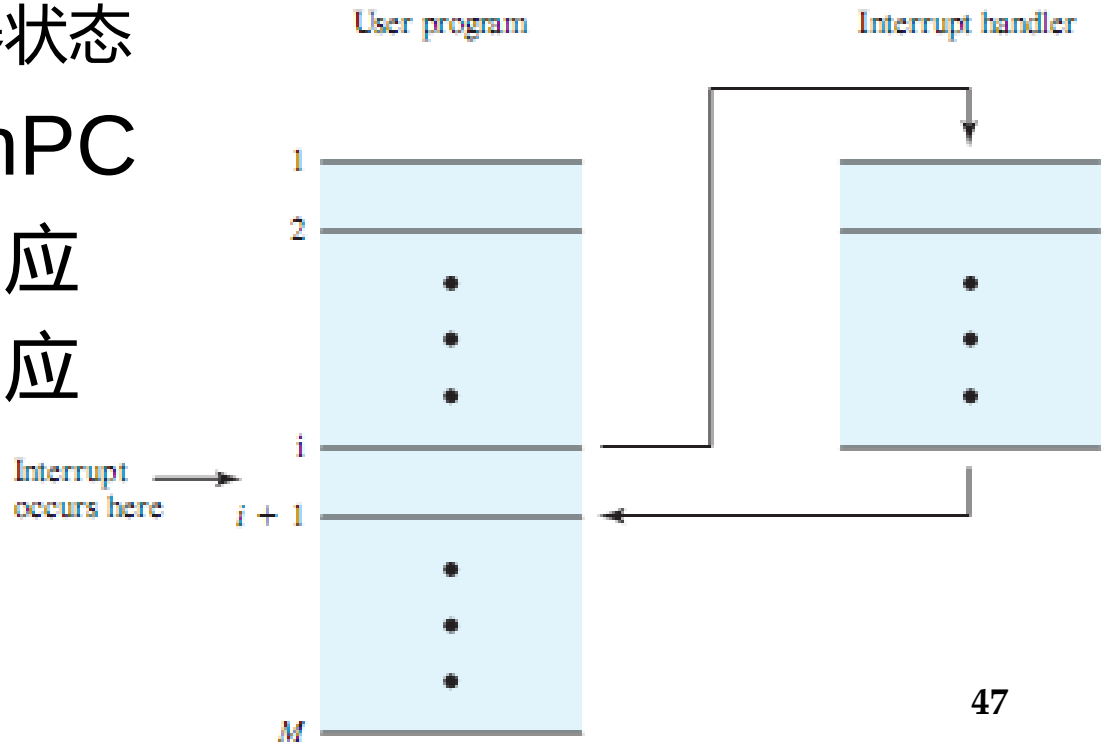
多周期的其他中断控制



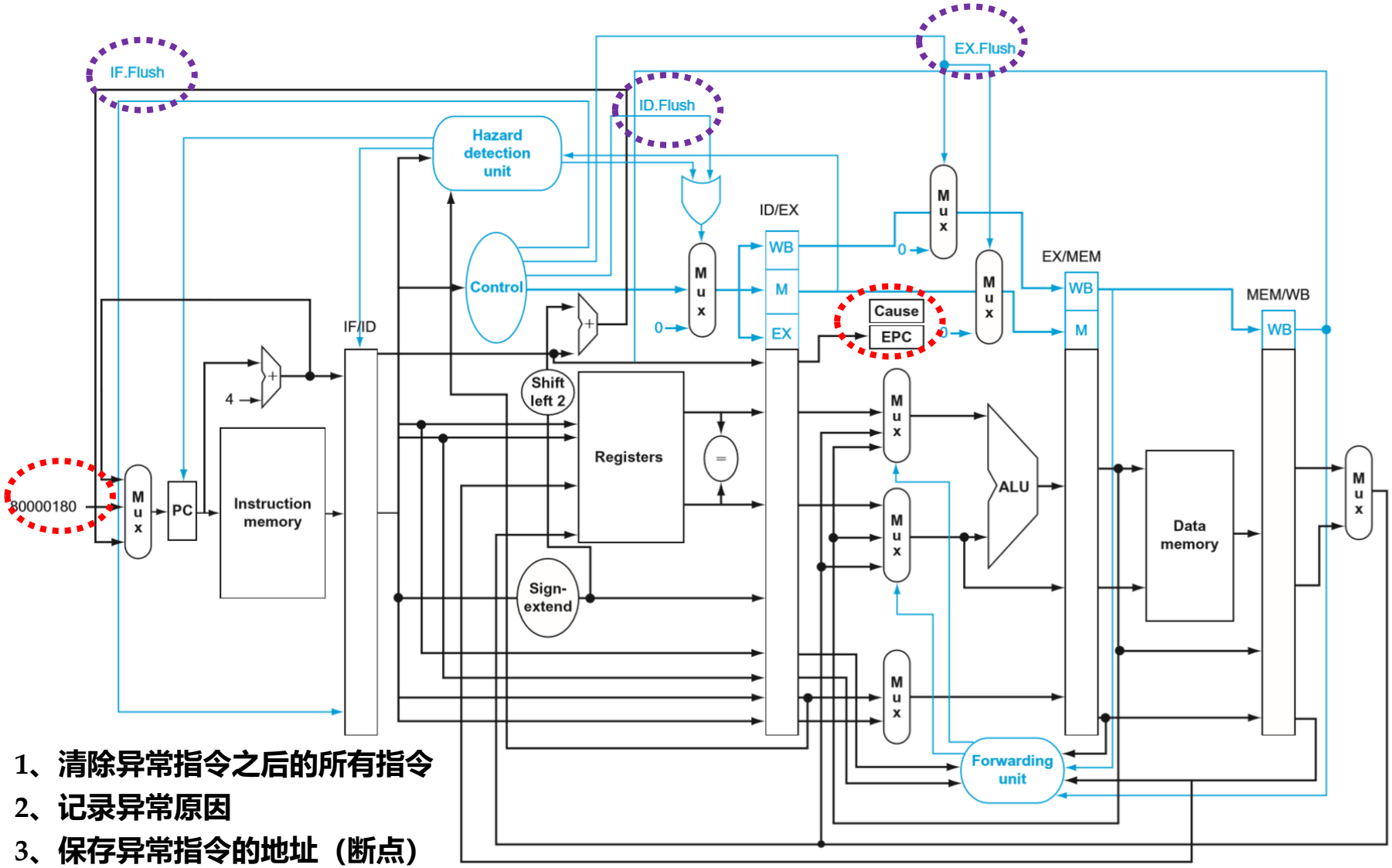
Check for interrupts before fetching next instruction

中断和异常操作语义特点（非流水线）

- 顺序语义
  - ✓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
  - ✓外部中断：异步响应
  - ✓指令异常：同步响应
- 现场简明
  - ✓Cause
  - ✓EPC



例子：在MIPS流水线中实现算术溢出异常（COD4）



1、清除异常指令之后的所有指令
2、记录异常原因
3、保存异常指令的地址（断点）
4、转到服务程序运行

具体指令的例子

Exception in a Pipelined Computer

Given this instruction sequence,

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

assume the instructions to be invoked on an exception begin like this:

```
80000180hex   sw     $26, 1000($0)
80000184hex   sw     $27, 1004($0)
...
```

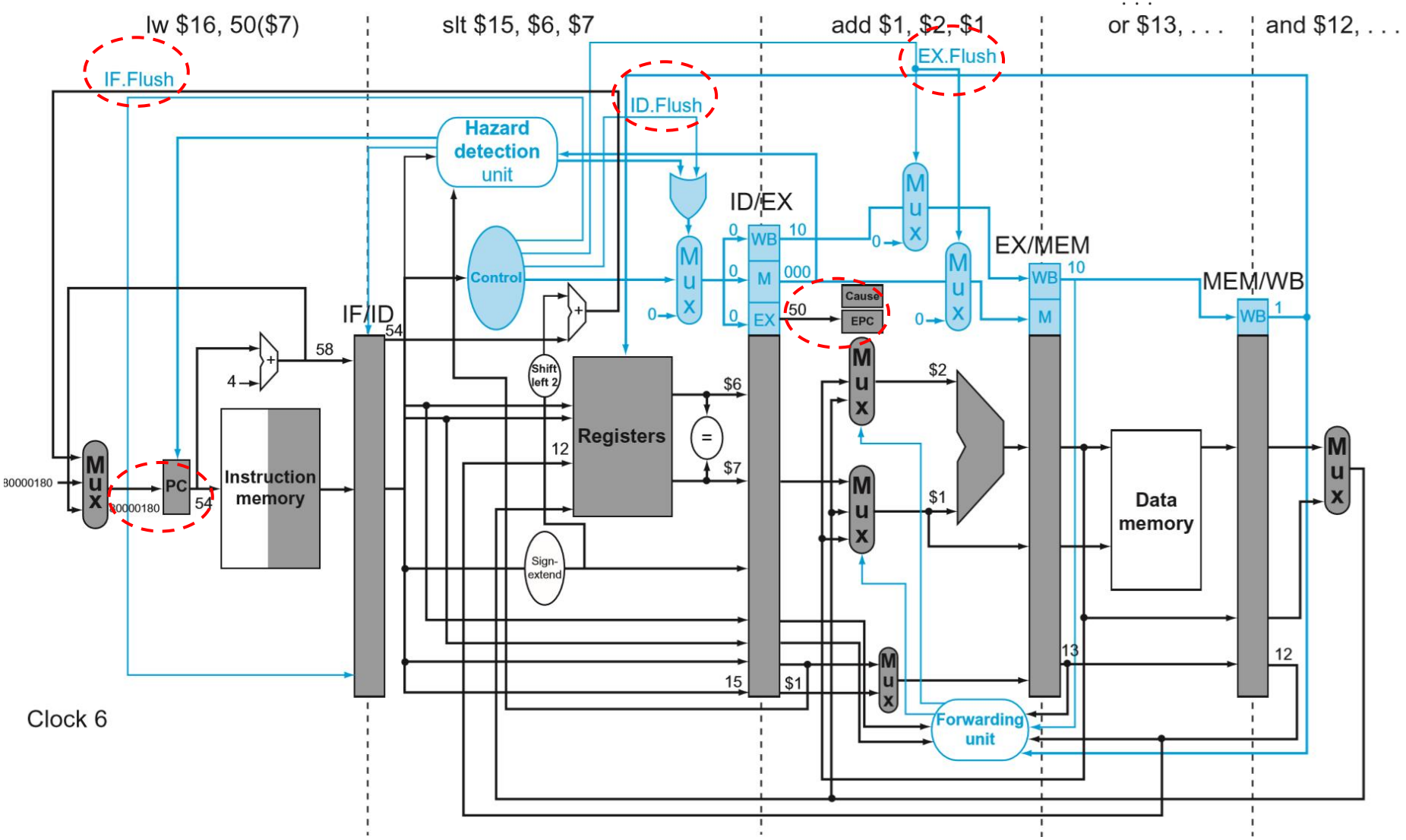
Show what happens in the pipeline if an overflow exception occurs in the `add` instruction.

Add在EX段

Exception in a Pipelined Computer

Given this instruction sequence,

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

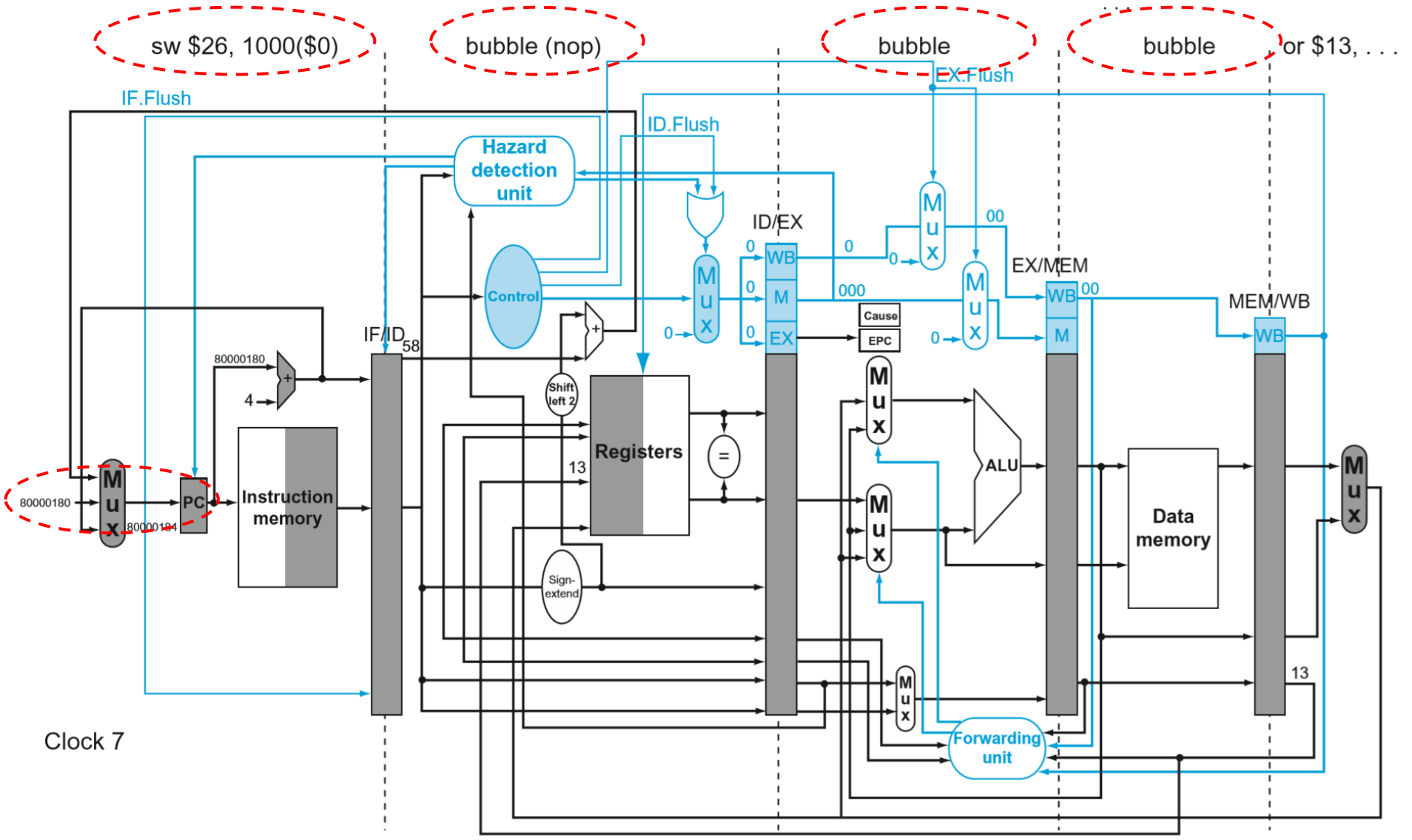


跳转到异常服务程序

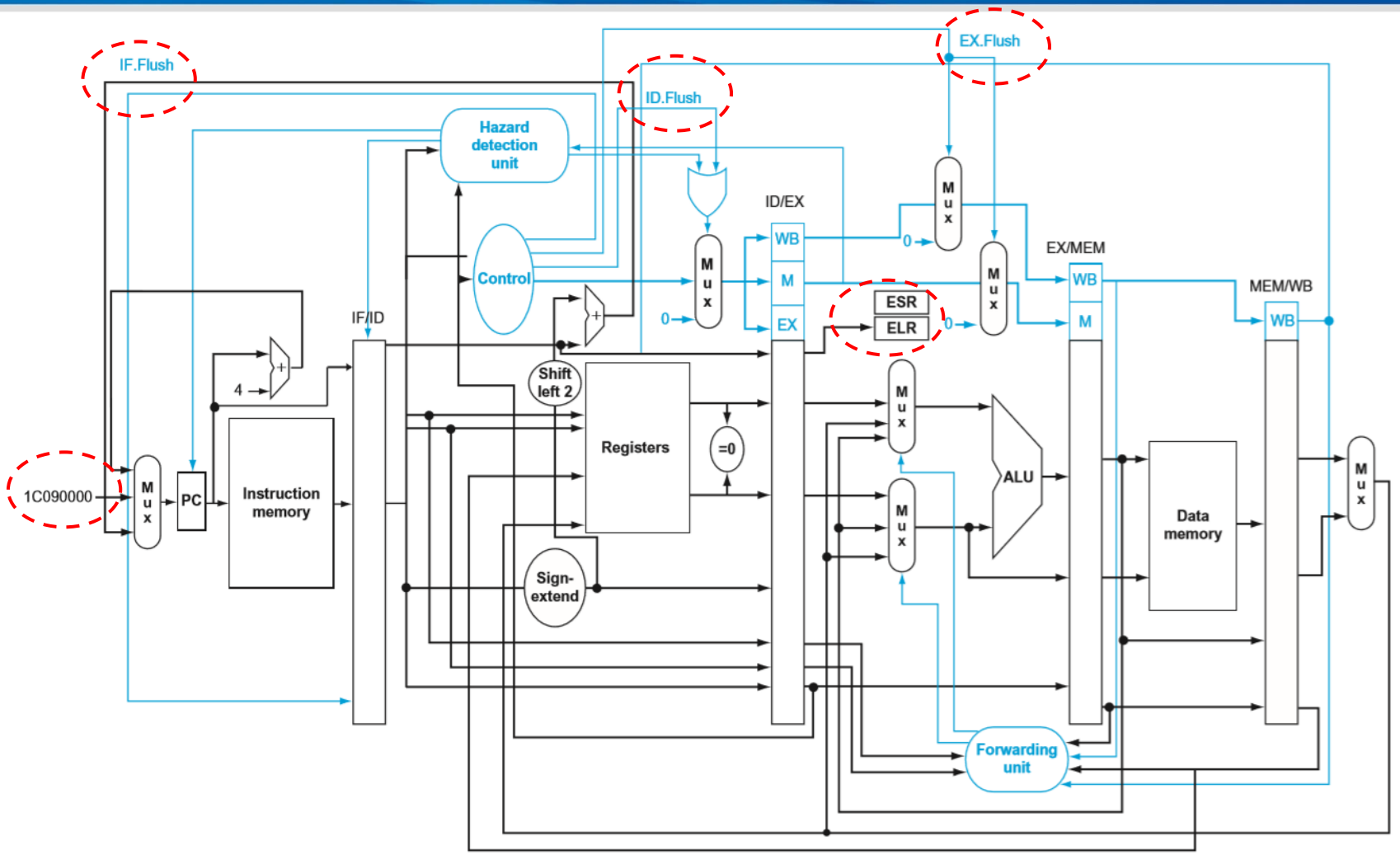
Exception in a Pipelined Computer

Given this instruction sequence,

```
40hex   sub   $11, $2, $4
44hex   and   $12, $2, $5
48hex   or    $13, $2, $6
4Chex   add   $1, $2, $1
50hex   slt   $15, $6, $7
54hex   lw    $16, 50($7)
```

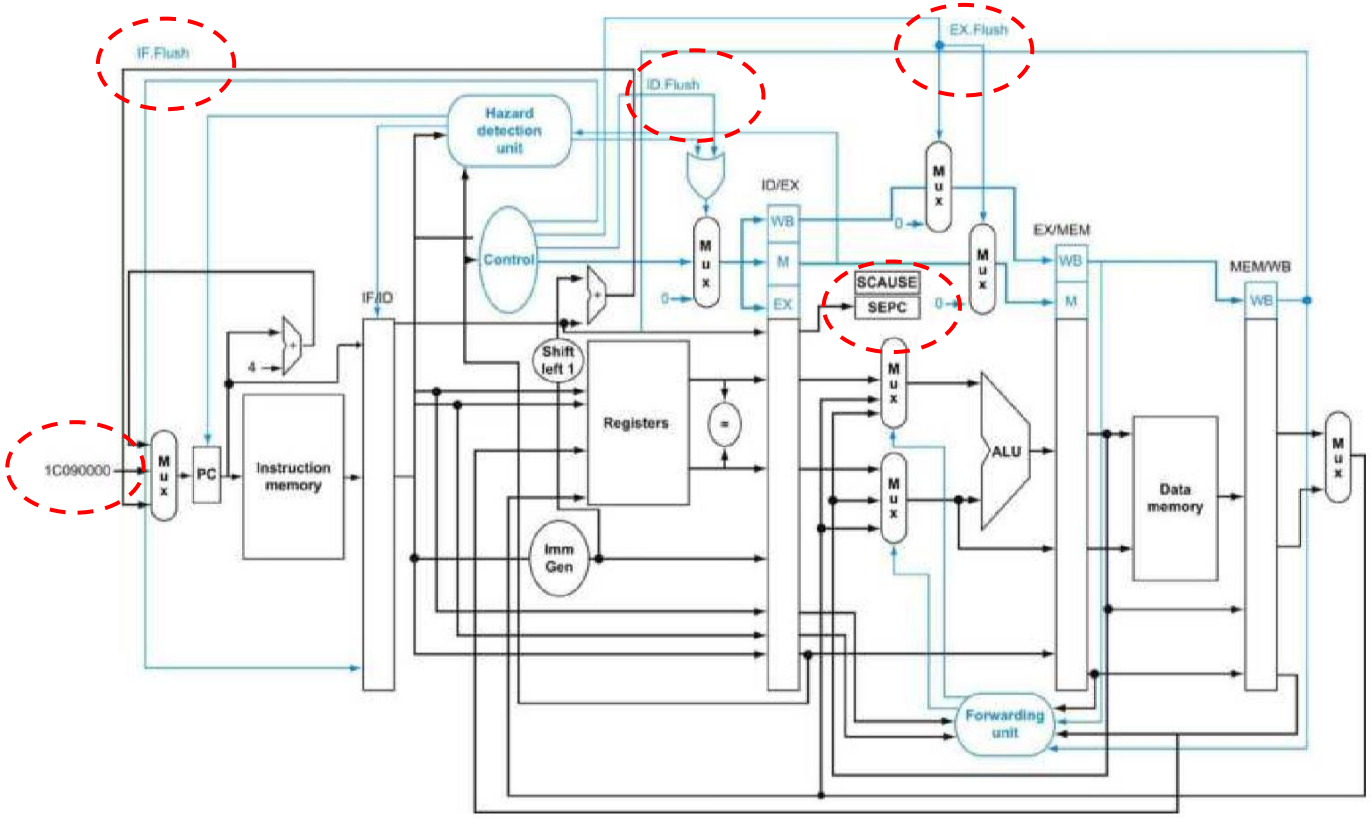


在ARM流水线中实现



Exception type	Exception vector address to be added to a Vector Table Base Register
Unknown Reason	$00\ 0000_{two}$
Floating-point arithmetic exception	$10\ 1100_{two}$
System Error (hardware malfunction)	$10\ 1111_{two}$

在RISC-V流水线中实现



Exception type	Exception vector address to be added to a Vector Table Base Register
Undefined instruction	$00\ 0100\ 0000_{two}$
System Error (hardware malfunction)	$01\ 1000\ 0000_{two}$

RISC-V处理异常的方式

CPU中异常的处理方式

- 由于异常是执行指令时同步发生，因此，在造成异常的指令之前执行的指令，均是有效的。
- 由于MIPS的高度流水体系结构，在引发异常的指令执行时，后面一条指令已经完成了读取和译码的预备工作。
- 当异常产生时，这些预备工作便被废弃。CPU从异常中返回时，再重新做读取和译码的工作。

《see mips run》

RISC-V异常的种类：30余种

- External events
  - ✓ 中断，读总线错
- Memory translation exceptions
  - ✓ 缺页，越界
- 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.）
  - ✓ 校验错
- System Calls and traps

Type of event	From where?	RISC-V terminology
System reset	External	Exception
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Either

RISC-V处理异常的方法

- 保存导致异常（或被中断）的指令的PC值
  - ✓ RISC-V使用SEPC（Supervisor Exception Program Counter，管理员异常程序计数器）
- 保存问题的表征
  - ✓ RISC-V使用SCAUSE（Supervisor Exception Cause Register，管理员异常原因寄存器）
  - ✓ 64位，但多数位没有用到：异常编码字段：2为未定义的操作码，12为硬件故障，…
- 跳转到处理程序
  - ✓ 假定程序位于0000 0000 1C09 0000hex

Type of event	From where?	RISC-V terminology
System reset	External	Exception
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Either

RISC-V异常/中断的5种属性

Exception type	Synchronous vs. asynchronous	User request vs. coerced	User maskable vs. nonmaskable	Within vs. between instructions	Resume vs. terminate
I/O device request	Asynchronous	Coerced	Nonmaskable	Between	Resume
Invoke operating system	Synchronous	User request	Nonmaskable	Between	Resume
Tracing instruction execution	Synchronous	User request	User maskable	Between	Resume
Breakpoint	Synchronous	User request	User maskable	Between	Resume
Integer arithmetic overflow	Synchronous	Coerced	User maskable	Within	Resume
Floating-point arithmetic overflow or underflow	Synchronous	Coerced	User maskable	Within	Resume
Page fault	Synchronous	Coerced	Nonmaskable	Within	Resume
Misaligned memory accesses	Synchronous	Coerced	User maskable	Within	Resume
Memory protection violations	Synchronous	Coerced	Nonmaskable	Within	Resume
Using undefined instructions	Synchronous	Coerced	Nonmaskable	Within	Terminate
Hardware malfunctions	Asynchronous	Coerced	Nonmaskable	Within	Terminate
Power failure	Asynchronous	Coerced	Nonmaskable	Within	Terminate

H&P CA图A.27

RISC-V处理异常的方法

- 向量化中断
  - ✓ 处理程序的地址由中断原因决定
- 向量表基地址寄存器加上异常向量地址：
  - ✓ 未定义的操作码： 00 0100 0000two
  - ✓ 硬件故障： 01 1000 0000two

Exception type	Exception vector address to be added to a Vector Table Base Register
Undefined instruction	$00\ 0100\ 0000_{two}$
System Error (hardware malfunction)	$01\ 1000\ 0000_{two}$

- 处理程序的功能为
  - ✓ 处理中断，
  - ✓ 或是跳转到实际的处理程序

RISC-V多周期与MIPS区别

Step	R-Type	lw/sw	beq/bne	j
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]) 访存指令PC+offset	If (A==B) then PC = ALUOut	PC = PC[31-28] || (IR[25-0]<<2
MEM	Reg[IR[15-11]] = ALUOut rd	MDR=Mem[ALUOut] Mem[ALUOut] = B		
WB		Reg[IR[20-16]] = MDR rd		

A=rs1 B=rs2
B指令PC+offset

跳转指令PC+offset

根据RISC-V与MIPS的多周期数据通路区别考虑异常处理方式?

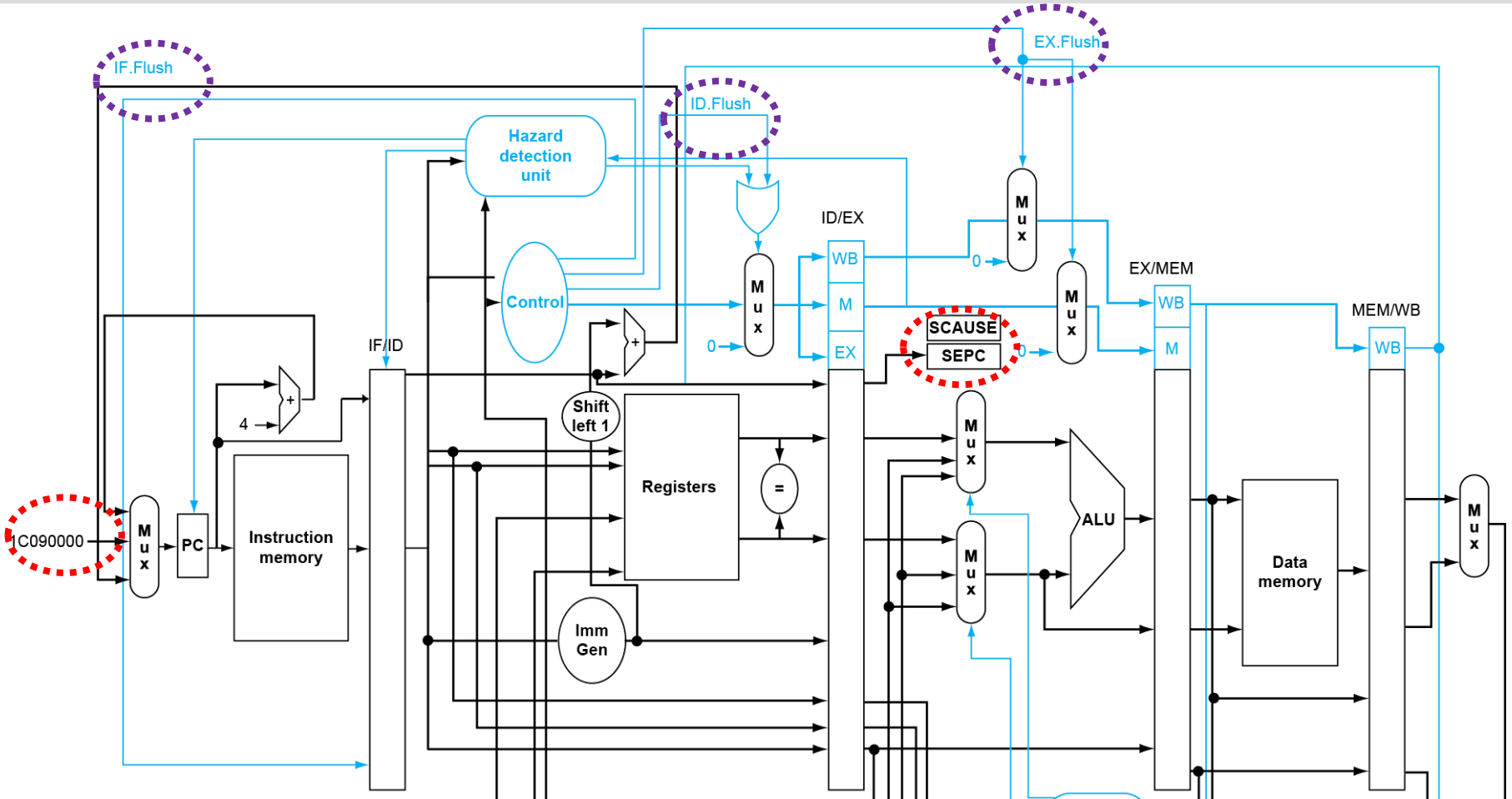
流水线中的异常

- 另一种形式的控制冒险
- 考虑add在EX级发生的故障

  add x1, x2, x1

  - ✓防止x1被错误赋值
  - ✓完成先前的指令
  - ✓清除add及之后的指令
  - ✓设置SEPC和SCAUSE寄存器的值
  - ✓把控制转移到处理程序
- 与**误预测的分支**类似
  - ✓用到的硬件多数相同

带异常处理的RISC-V流水线



1、清除异常指令之后的所有指令
2、记录异常原因
3、保存异常指令的地址（断点）
4、转到服务程序运行

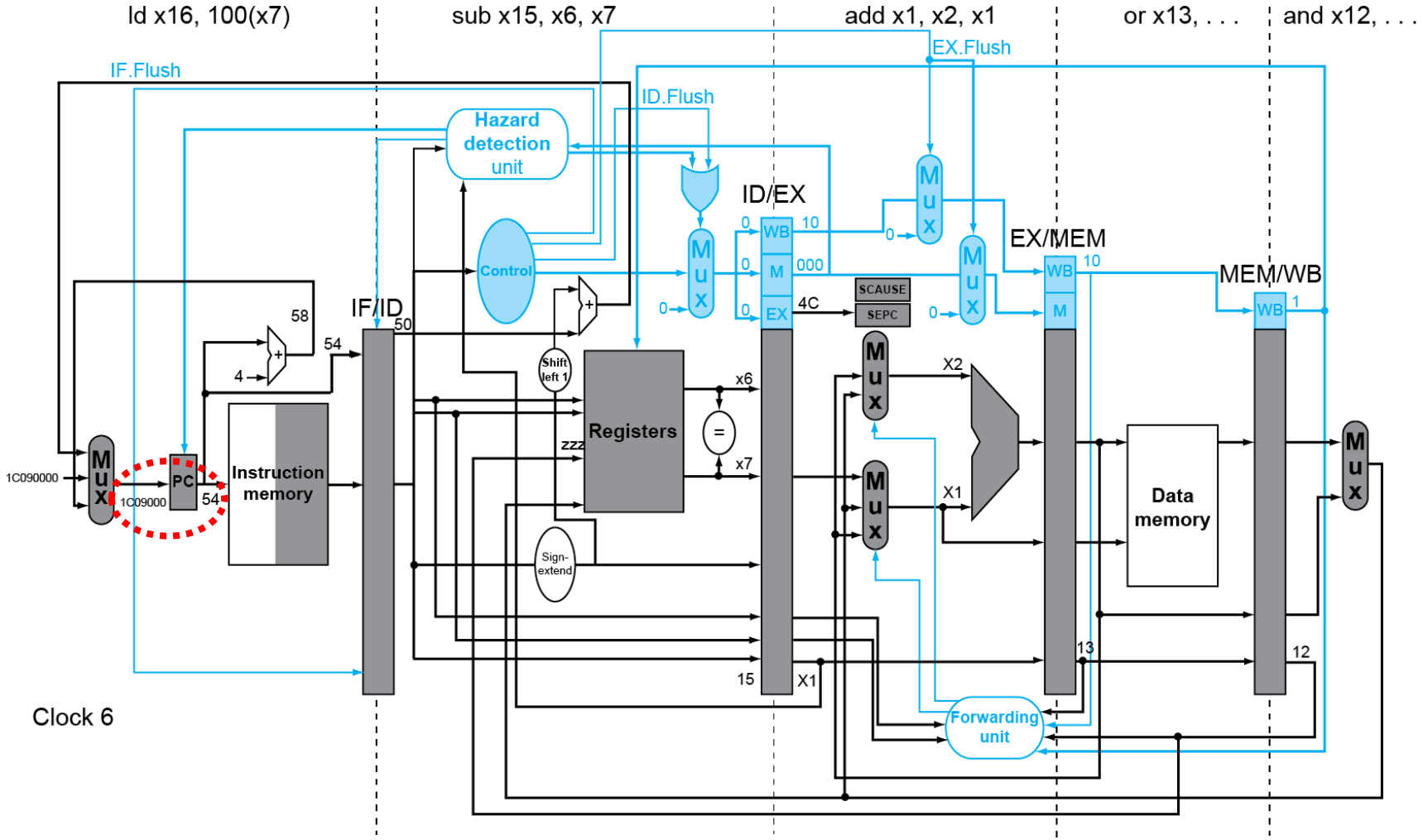
例子

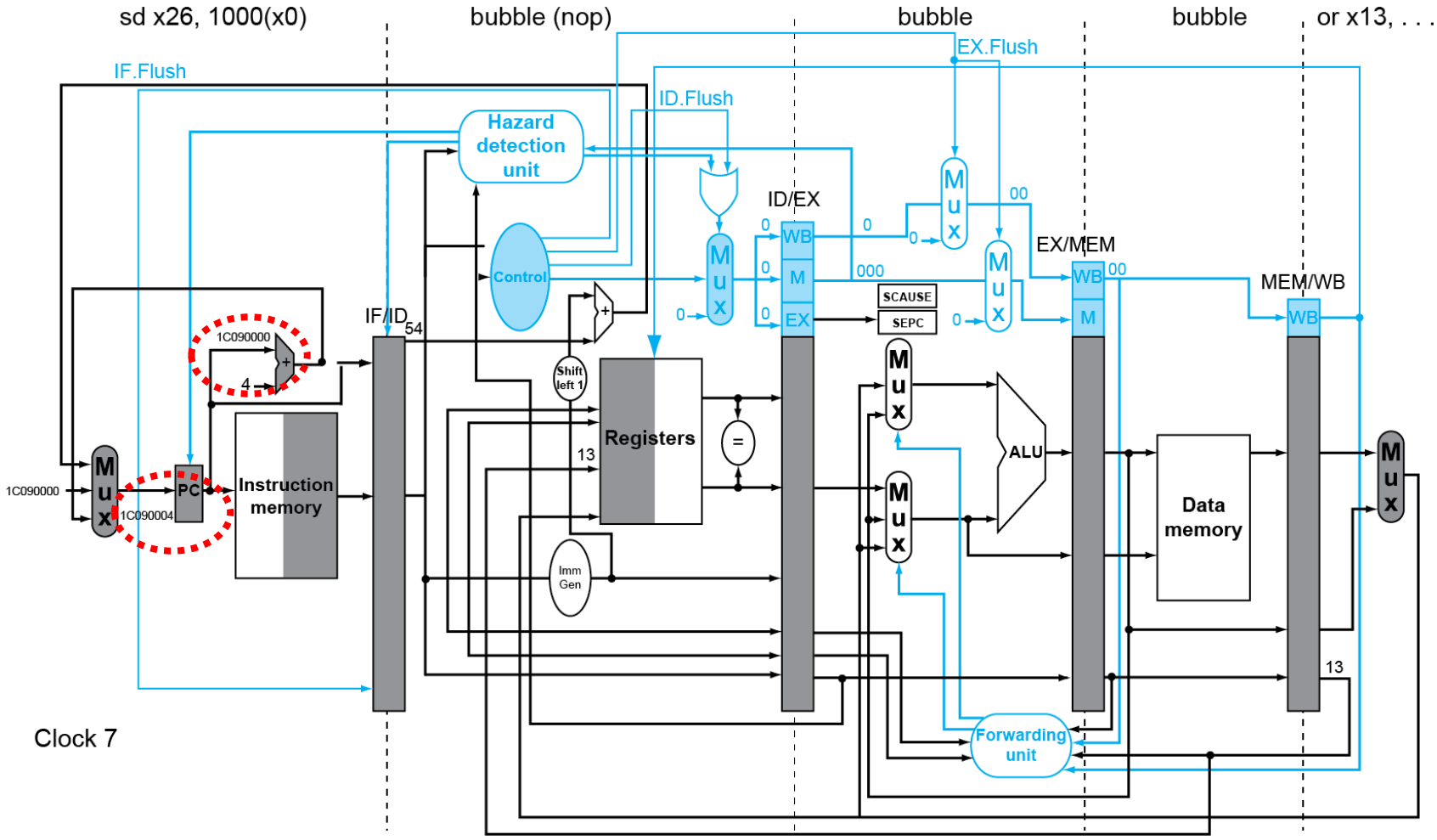
- Exception on add in

```
40    sub   x11, x2, x4
44    and   x12, x2, x5
48    orr   x13, x2, x6
4c    add   x1,  x2, x1
50    sub   x15, x6, x7
54    ld    x16, 100(x7)
…
```

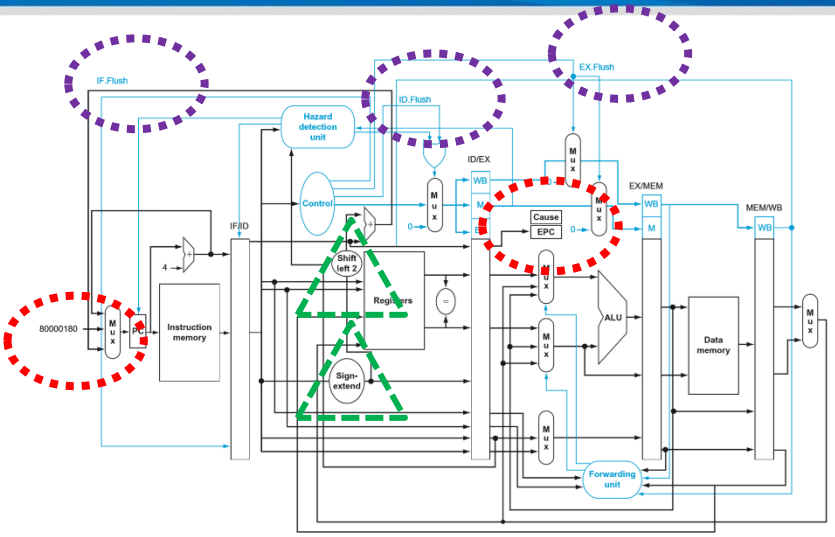
- Handler

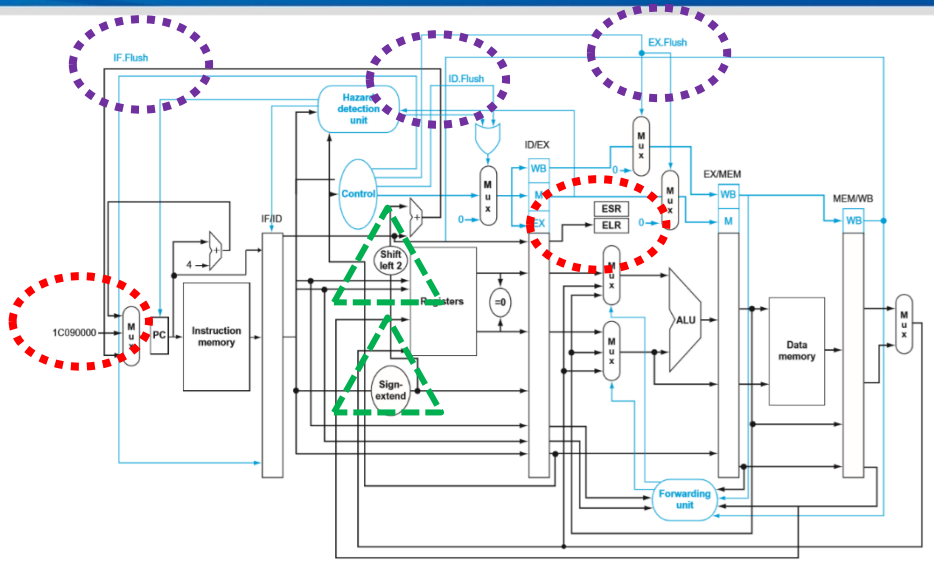
```
1c090000  sd  x26, 1000(x10)
1c090004   sd  x27, 1008(x10)
…
```

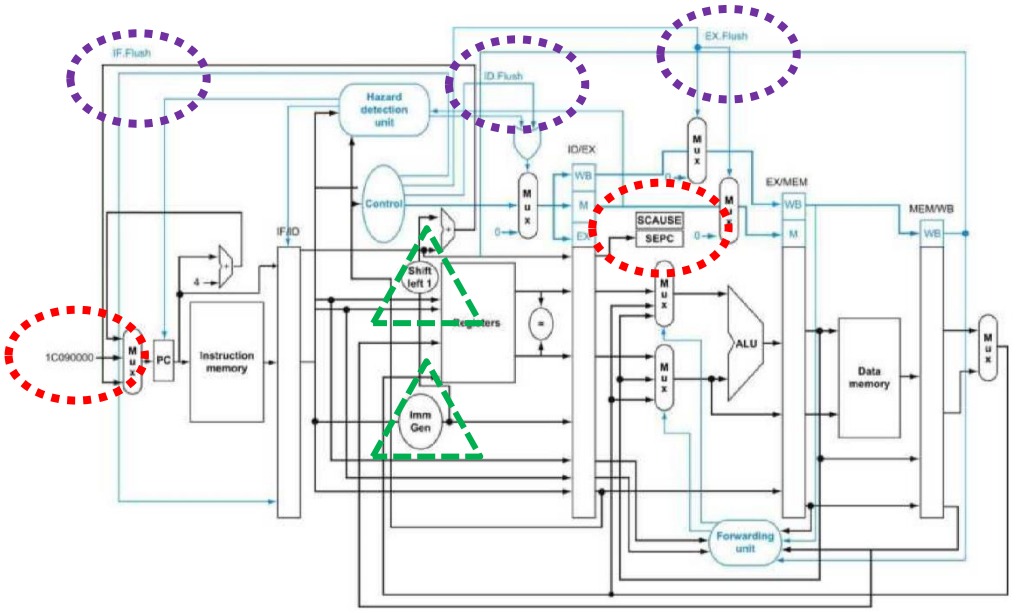

ld x16, 100(x7)
sub x15, x6, x7
add x1, x2, x1
or x13, . . .
and x12, . . .
IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
IF/ID
WB
M
EX
SCAUSE
SEPC
4C
Shift left 1
Registers
Sign-extend
Instruction memory
Data memory
Forwarding unit
PC
1C090000
Clock 6


sd x26, 1000(x0)
bubble (nop)
bubble
bubble
or x13, . . .
IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
SCAUSE
SEPC
000
00
IF/ID
54
1C090000
4
1C090004
PC
Instruction memory
Shift left 1
13
Registers
=
Imm Gen
ALU
Data memory
Forwarding unit
Mux
Clock 7

Summary: MIPS-ARM-RISCV





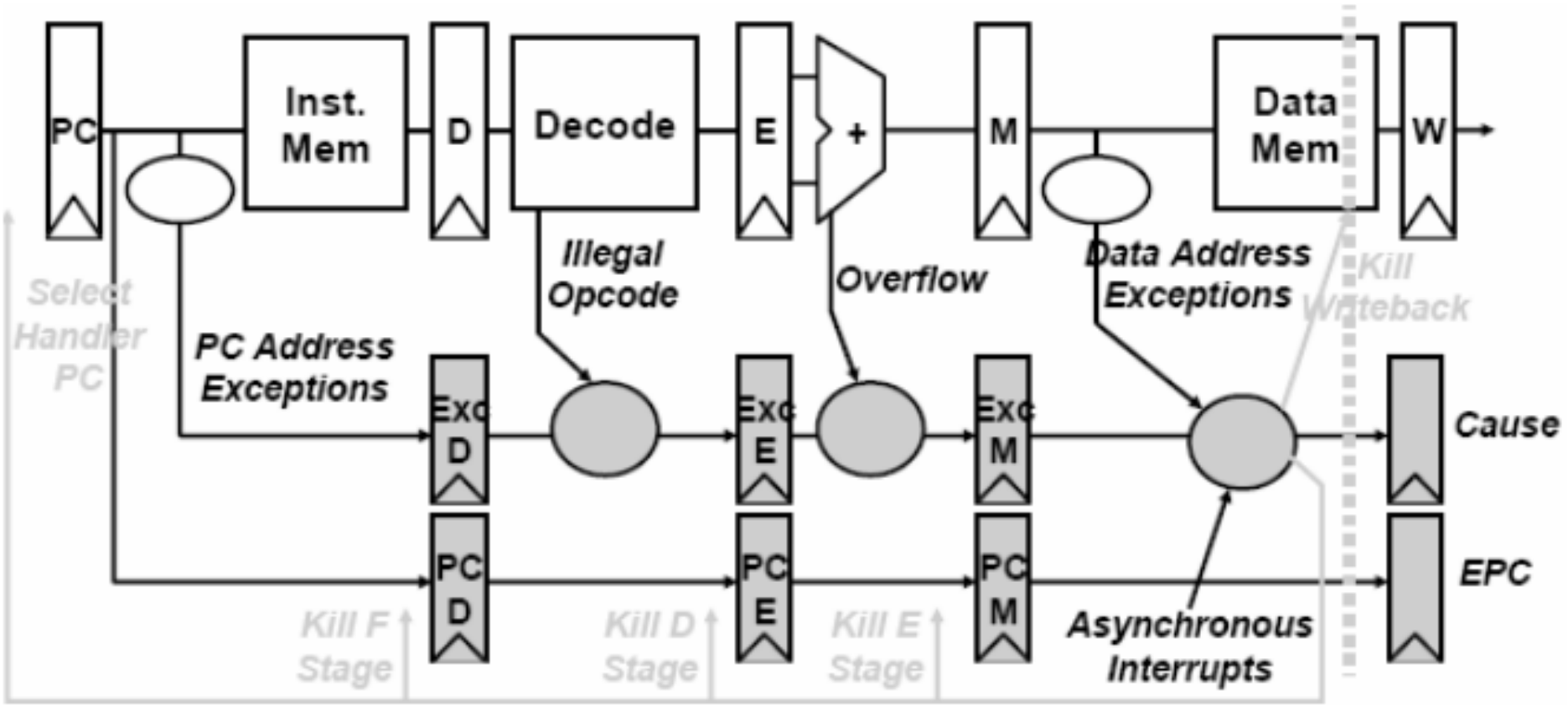


1、清除异常指令之后的所有指令
2、记录异常原因
3、保存异常指令的地址（断点）
4、转到服务程序运行

多重异常与非精确处理

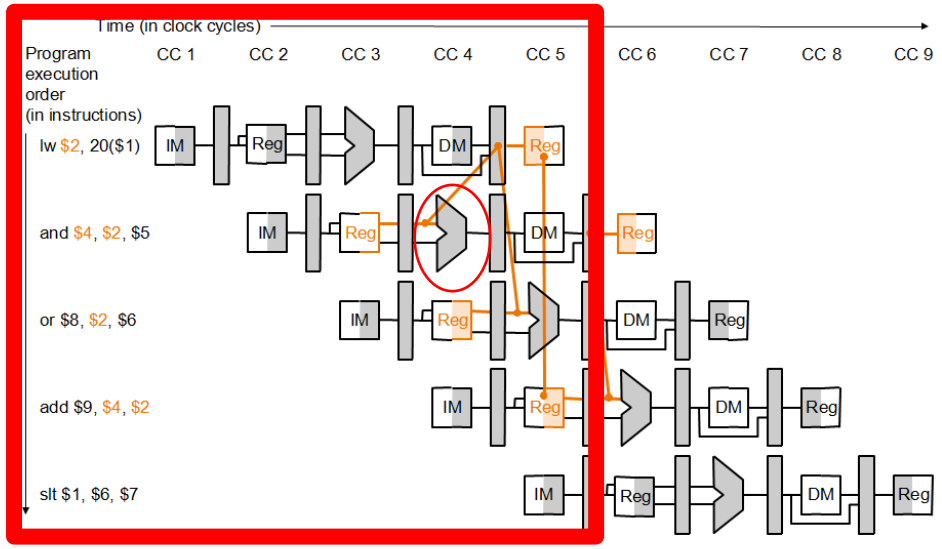
流水线中的异常与中断

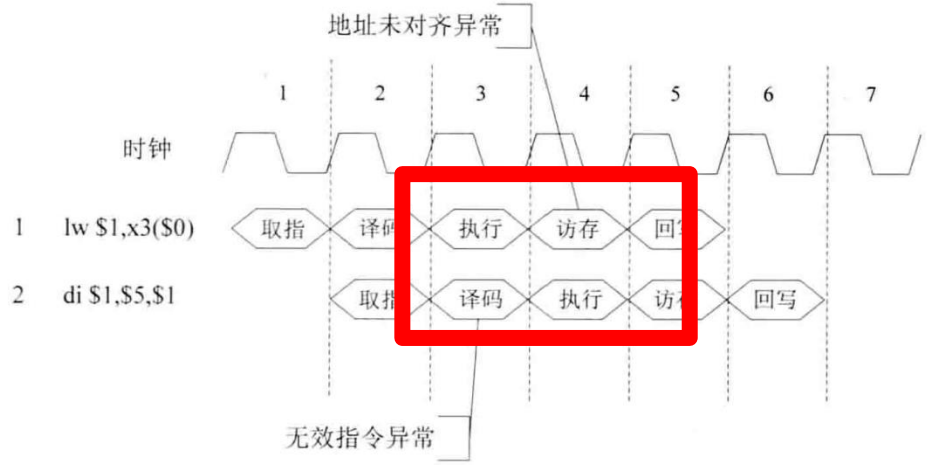
- 多个流水段，多条指令，多种异常并发
  - ✓异常：访存-地址错（缺页越界），译码-非法指令，ALU溢出
  - ✓中断



流水线异常处理的挑战

- “顺序执行”只是一种逻辑关系：断点和状态难以确定
  - ✓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0标志
  - ✓异常发生顺序与指令执行顺序不一定相同
    - 如di指令在ID段CC3发生异常，但LW指令在MEM段CC4异常
  - ✓转移指令和分支预测给异常处理带来了麻烦

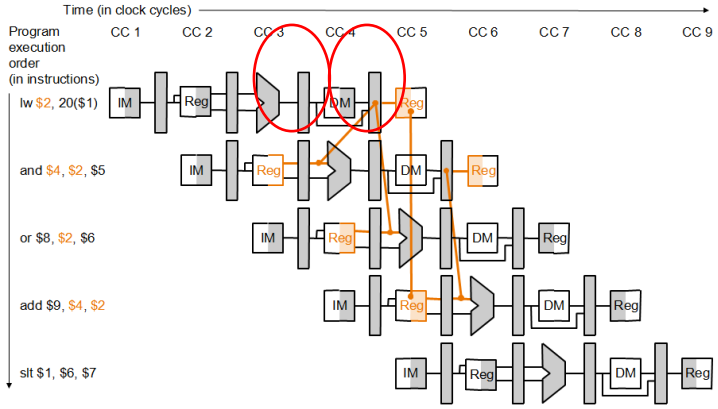




解决方案：非精确处理、精确处理

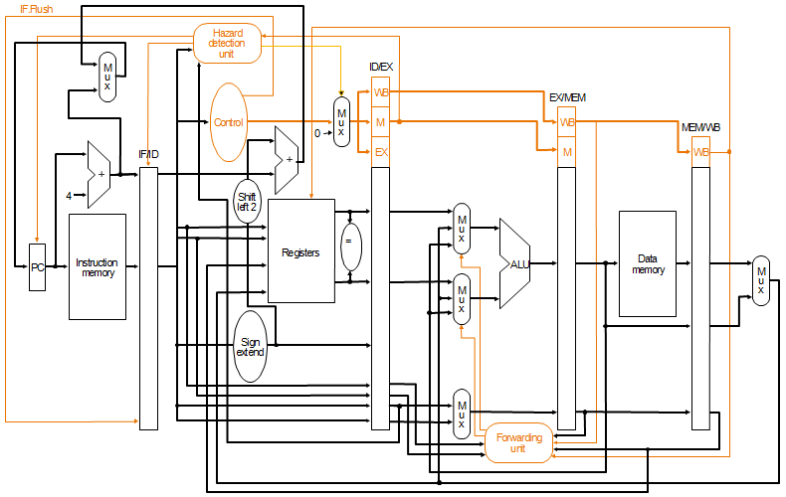
思路1：非精确处理异常的方法

- 非精确方案1：允许已进入流水线中的指令执行完再转去执行中断处理
  - ✓断点：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，断点为最后进入流水线的那条指令的地址(非精确)
    - 非精确（可能不是“当前指令”）
    - 可变（不同段发生异常，EPC增量不同）
- 优点：硬件比较简单



非精确异常

- 非精确方案2：将异常指令的后续指令排空，COD
- 处理：将异常视为一种控制相关，工作如下：
  - ✓暂停指令流中导致异常的指令
  - ✓执行完异常指令之前的所有指令
  - ✓清除异常指令之后的所有指令
  - ✓记录异常原因
  - ✓保存断点
  - ✓转异常处理程序
- 不允许后续指令继续执行



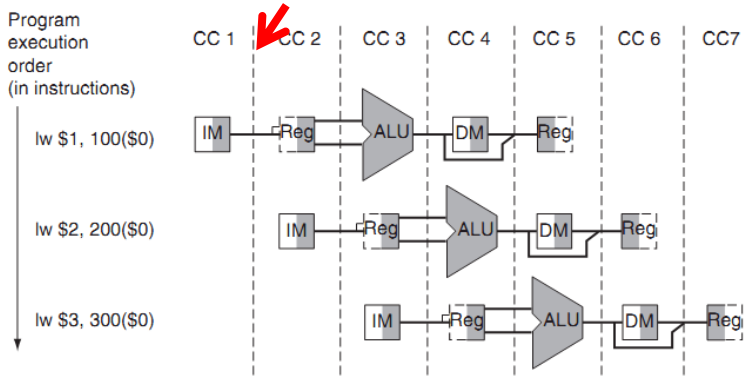
非精确异常的问题

◘缺点

- ✓ 异常响应时间较长
- ✓ 如果等进入流水线的指令执行结束，可能会导致程序出错
  - – i: ADD R1, R2; (R1)+(R2) ->R1，如果此时溢出?
  - – i+1: MUL R3, R1; (R3)×(R1) ->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。

◘如何非精确处理?

◘如何处理多个异常?

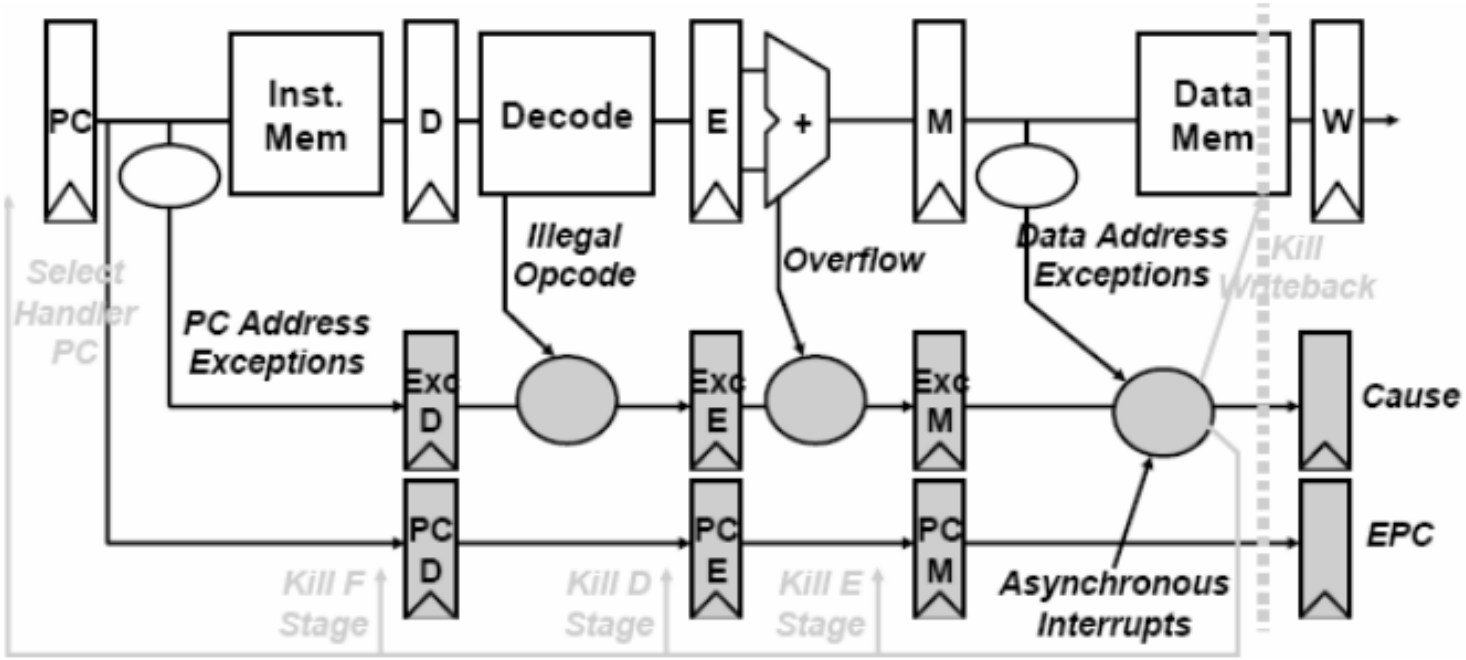


思路2：如何实现精确异常

- 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、PSW、流水段寄存器（含各段的控制寄存器）！
- 例如：采用提交点技术实现“精确”异常
  - ✓提交点：M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

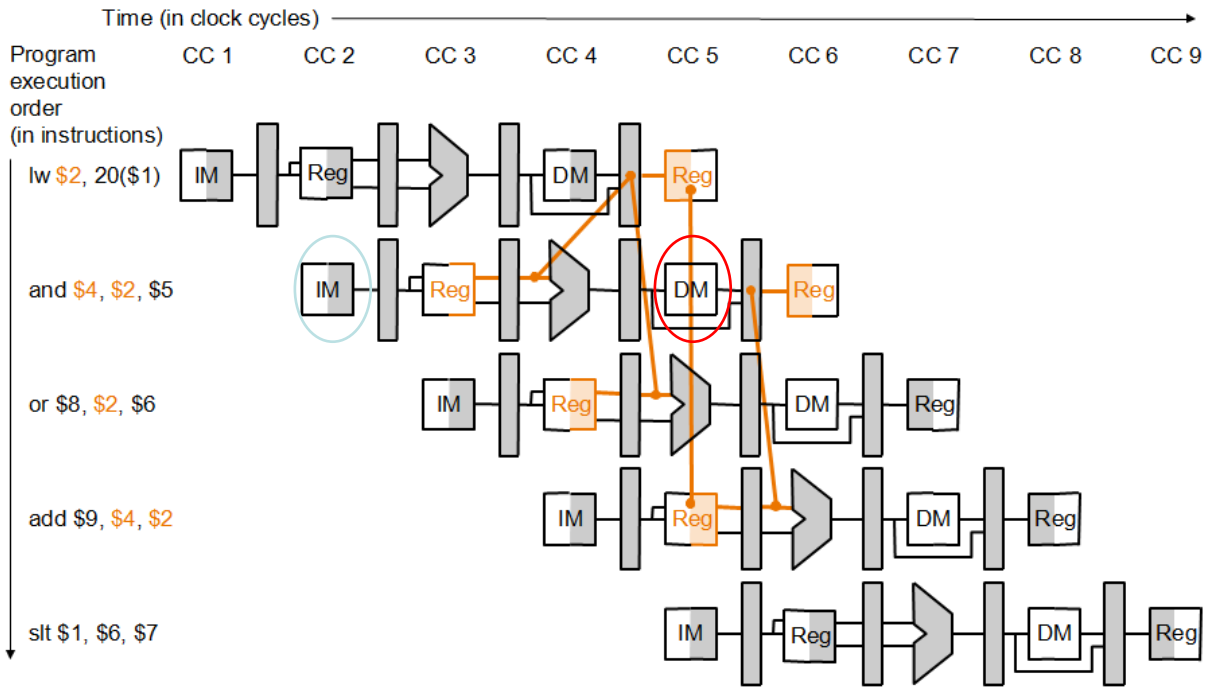
各段产生的异常及处理：MIPS的策略

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



为何是M？EXE或WB是否可以？

- 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（保证）
  - ✓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



异常响应的顺序

MIPS优先级从高到低

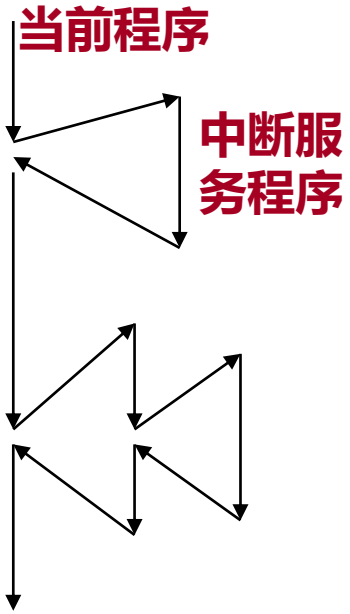
1. Reset (highest priority)
2. Soft Reset
3. Nonmaskable Interrupt (NMI)
4. Address error --Instruction fetch
5. TLB refill--Instruction fetch
6. TLB invalid--Instruction fetch
7. Cache error --Instruction fetch
8. Bus error --Instruction fetch
9. Watch - Instruction Fetch
10. Integer overflow, Trap, System Call, Breakpoint, Reserved Instruction, Coprocessor Unus-able, or Floating-Point Exception Address error--Data access
11. TLB refill --Data access
12. TLB invalid --Data access
13. TLB modified--Data write
14. Cache error --Data access
15. Watch - Data access
16. Virtual Coherency - Data access
17. Bus error -- Data access
18. Interrupt (lowest priority)

Arm优先级从高到低：

(1) Reset (highest priority)
(2) Data abort
(3) FIQ（快速中断）
(4) IRQ
(5) Prefetch abort
(6) 未定义指令, Software interrupt

总结：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，在它们执行完成后再恢复被中断程序的执行。
  - ✓ 中断源：外部中断、内部中断
    - 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
  - ✓ 中断嵌套（屏蔽字）



异常的实现

- 多周期实现异常（以MIPS为例）
  - ✓ 主要过程：判断异常，保存现场，跳转执行
  - ✓ 数据通路
  - ✓ RTL代码
  - ✓ 状态机
- 流水线实现异常（MIPS、RISC-V）
  - ✓ 非精确实现：判断异常，保存现场，清空指令，跳转执行
  - ✓ 精确实现：判断异常，暂存错误，到点提交，保存现场，清空指令，跳转执行
- 多周期、精确、非精确实现的区别

思考与作业

- 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
- 精确中断，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、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（交）：COD RISC-V 4.16 4~5、4.22、4.25
- 作业（不交）唐（8.23、8.24）、

练习

- 假设如下指令序列
  - ✓ 采用多周期执行方式和流水线非精确方式，在第几个周期检测出异常？（LW互锁=1）80000184 SW $27, 1004指令何时取指?

```
LOOP:   LW $1, 40($10) ；取$1
        ADD $2,$3,$1    ；加法
        SW $2, 40($10)；存
        SUB   $9,$9,$11          ；迭代器
        XXX $5,$7          ；异常-未定义指令（ID段检测）
        BNE   LOOP,$9,$0         ；分支指令

异常指令:   80000180  SW $26,1000（$0）
        80000184 SW $27, 1004（$0）
        …..
```

联系方式

Acknowledgements:
This slides contains materials from following lectures:
- **Computer Architecture (NUDT)**
- **CS 152 CS 252 (UC Berkeley)**

Research Area：
- 基于分布式系统, GPU，FPGA的神经网络、图计算加速
- 人工智能和深度学习（寒武纪）芯片及智能计算机

Contact：
- 中国科学技术大学计算机学院
- 嵌入式系统实验室（西活北一楼）
- 高效能智能计算实验室 （中科大苏州研究院）

cswang@ustc.edu.cn
http://staff.ustc.edu.cn/~cswang

"study the past if you would define the future."

by Confucius