异常与中断

异常并不意外！——llxx☺

llxx@ustc.edu.cn

程序如何执行的？用到哪些设备？

```
/*---sum.c---*/
int sum(int a[ ], unsigned len)
{
    int   i，  sum = 0;
    for   (i = 0; i <= len–1; i++)
          sum += a[i];
    return sum;
}

/*---main.c---*/
int main()
{
    int   a[1]={100};
    int   s;
    s=sum(a,0);
    printf(“%d”,s);
}
```

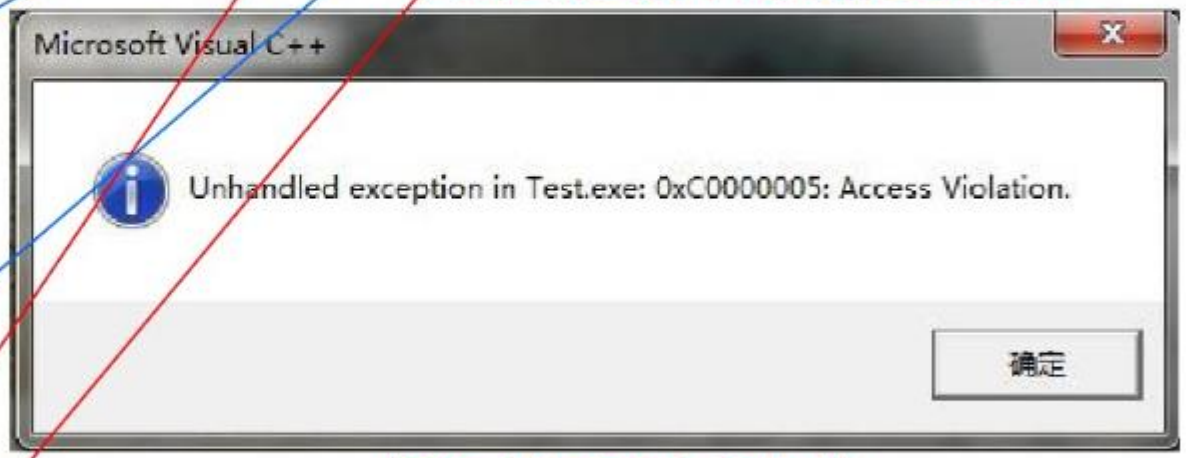
数据的表示

数据的运算

各类语句的转换与表示(指令)

各类复杂数据类型的转换表示

过程（函数）调用的转换表示



链接（linker）和加载

程序执行（存储器访问）

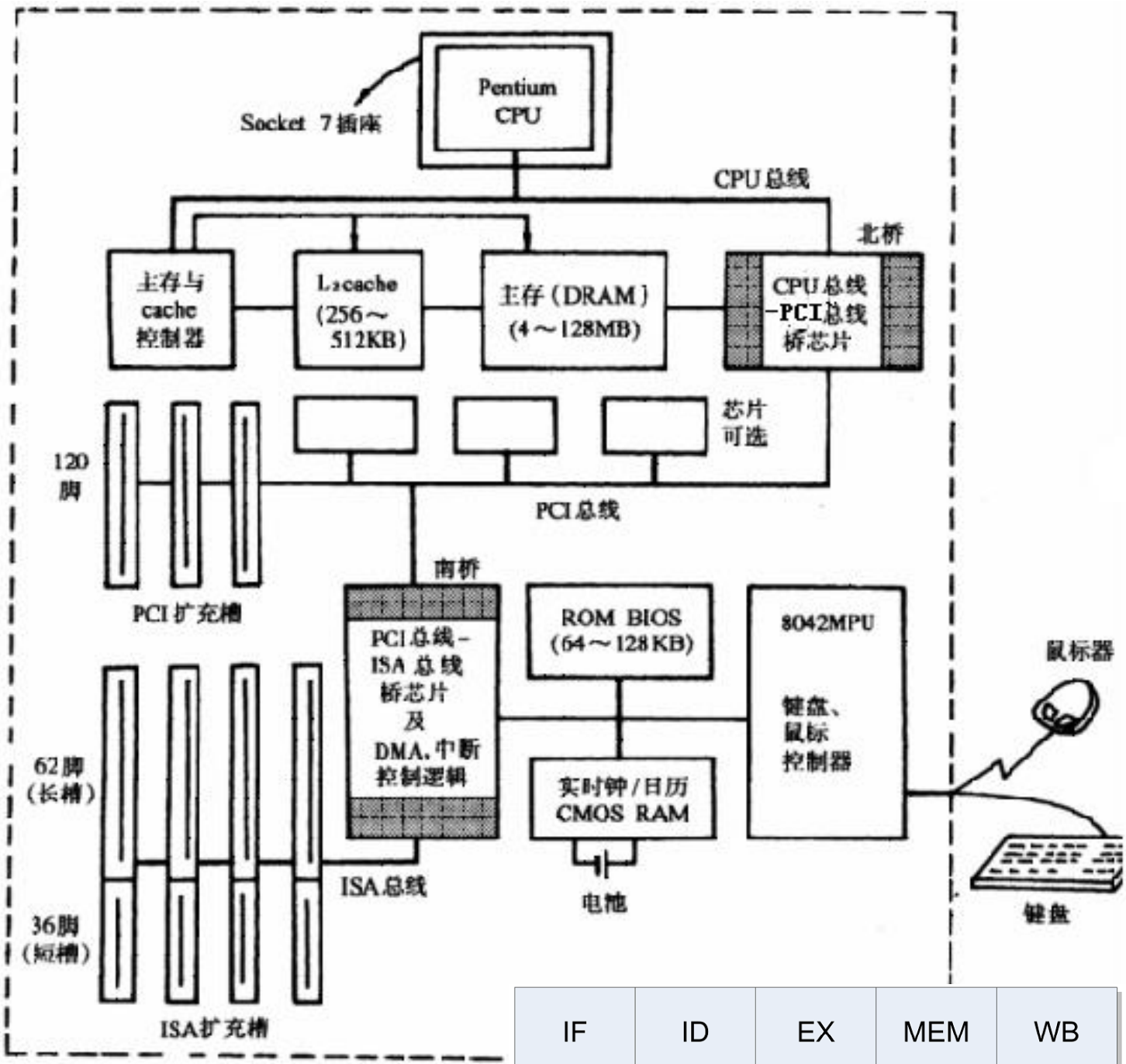
异常和中断处理

输入输出(I/O)

- C语言计算机？

IF	ID	EX	MEM	WB

输入输出

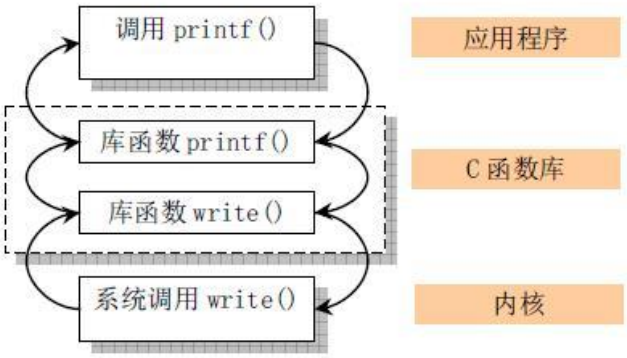


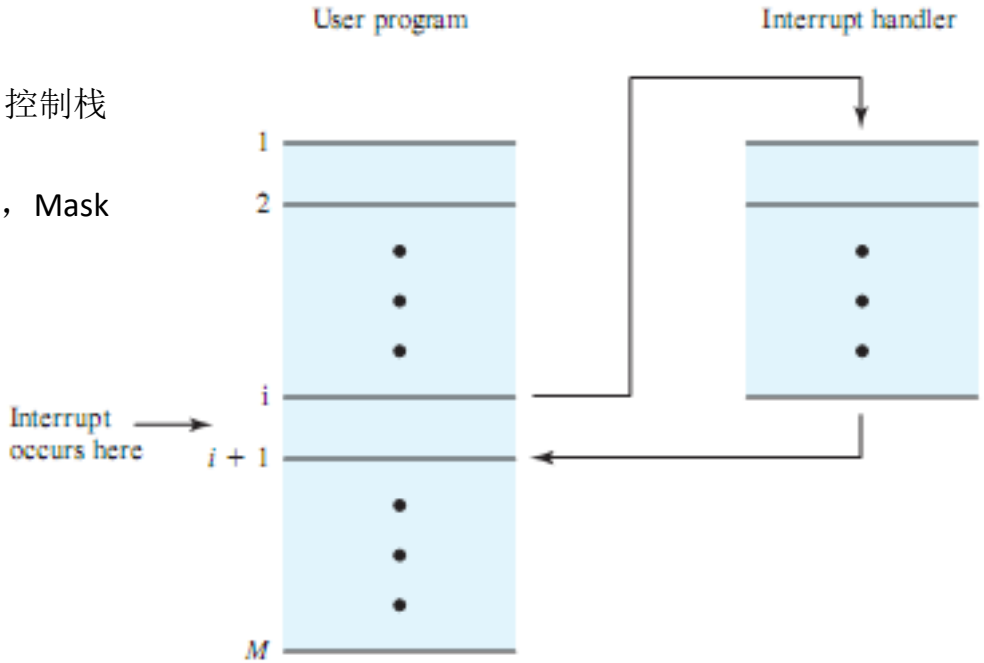
唐图10.19

IF	ID	EX	MEM	WB

内容提要

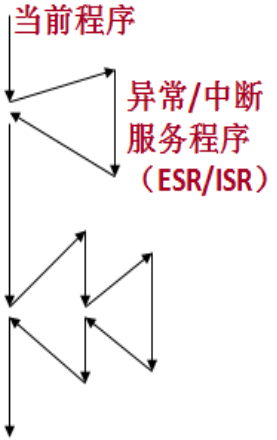
- 异常与中断的基本概念，4.9
  - 功能：处理意外事件，I/O
    - 指令异常，系统服务（syscall）
    - 外部中断
  - 时序：程序序（进程序）语义
    - 指令周期：同步，异步（中断周期）
  - 属性：5种
- 异常与中断响应的基本过程
  - 机构
    - ESR/ISR入口（向量式/非向量式），控制栈
    - 异常：EPC，Cause，Vector
    - 中断：EPC，Cause，Vector，Enable，Mask
  - OS：ESR/ISR
  - 约定：软硬件接口
- 异常与中断响应过程控制
  - 单周期
  - 多周期：中断周期
  - 流水线：精确，非精确





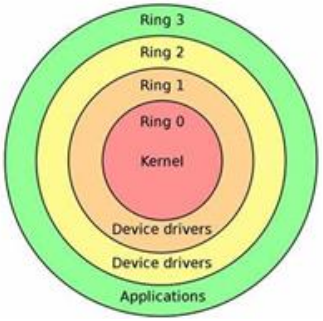
Terms：exceptions、interrupts、traps

- 改变正常指令执行流（Transfer of Control）的意外事件。
  - 暂停当前程序的执行，转而执行ESR/ISR程序；
  - ESR/ISR执行完成后，被中断的程序恢复执行。
  - 与过程调用的区别：发生场景（程序内/外），系统状态切换
- RISC：MIPS/ARM/RV统称exceptions，含
  - 内部事件：异常（例外）
    - 指令异常：非法指令、算术溢出/除零、访存缺页
    - 系统调用（syscall，软中断/陷阱）：服务、断点break
  - 外部事件：中断Interrupts，硬中断
    - I/O（键盘，可屏蔽），硬件故障（存储器，不可屏蔽）



p224

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



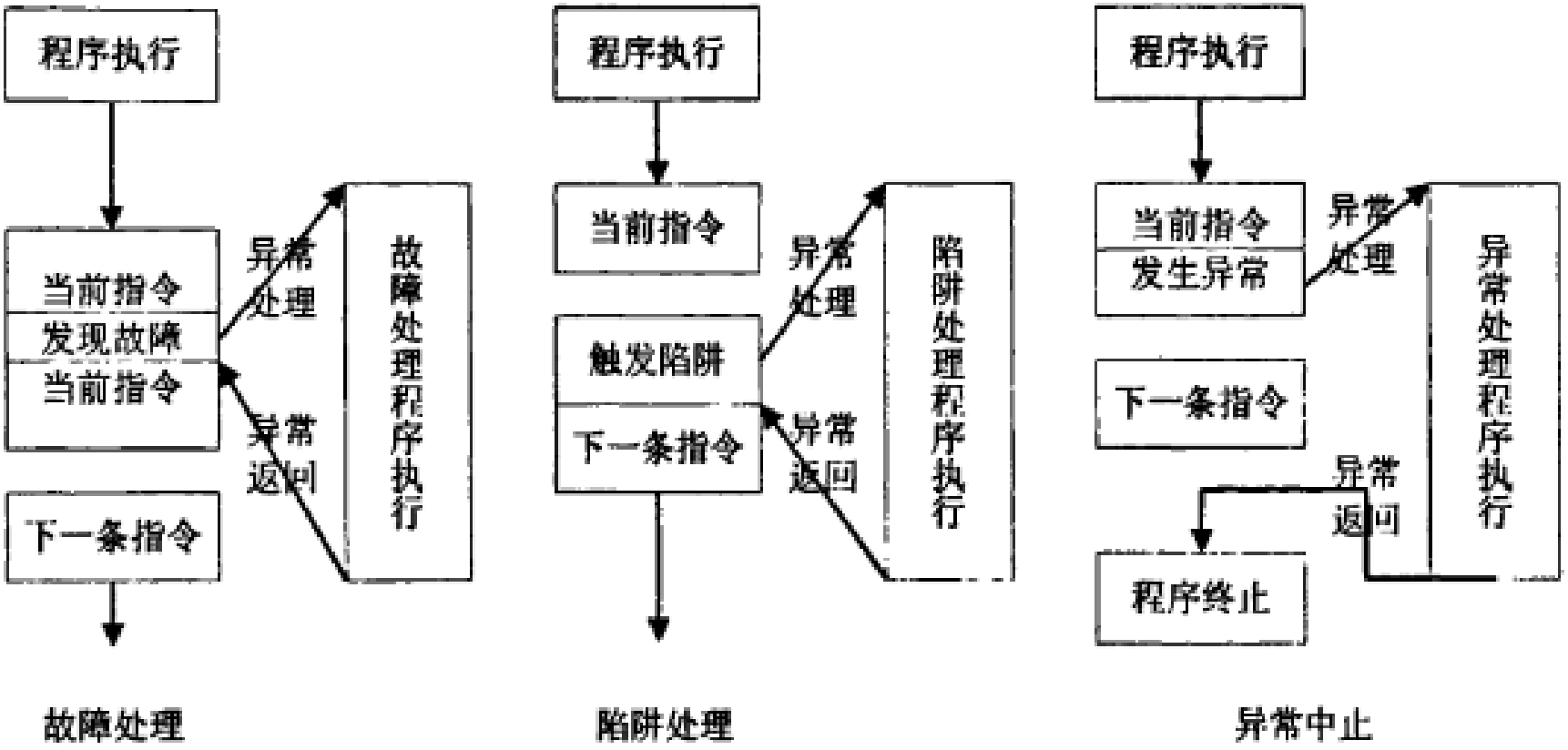
RISC异常/中断的属性：5种

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

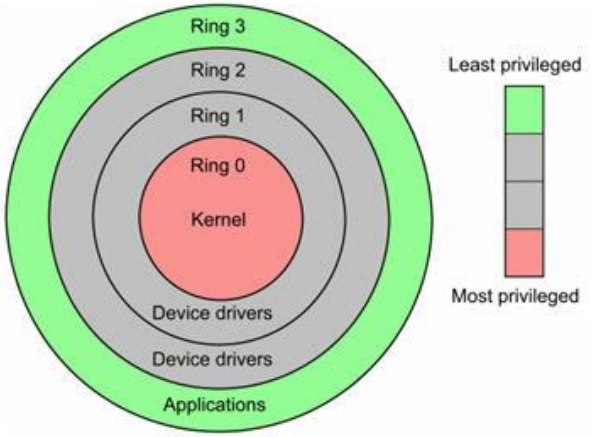
H&P CA图A.27

指令异常响应模式

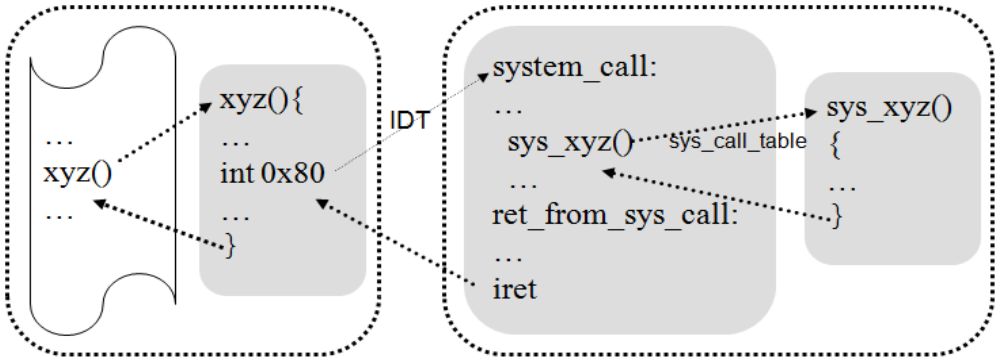
- 可恢复异常：访存指令缺页，OS处理后重新执行当前指令，如图“故障”
- 不可恢复异常：当前异常无法修复，交用户处理，如图“异常中止”
- 陷阱（软中断/系统调用）：系统调用指令触发请求，返回下一条指令



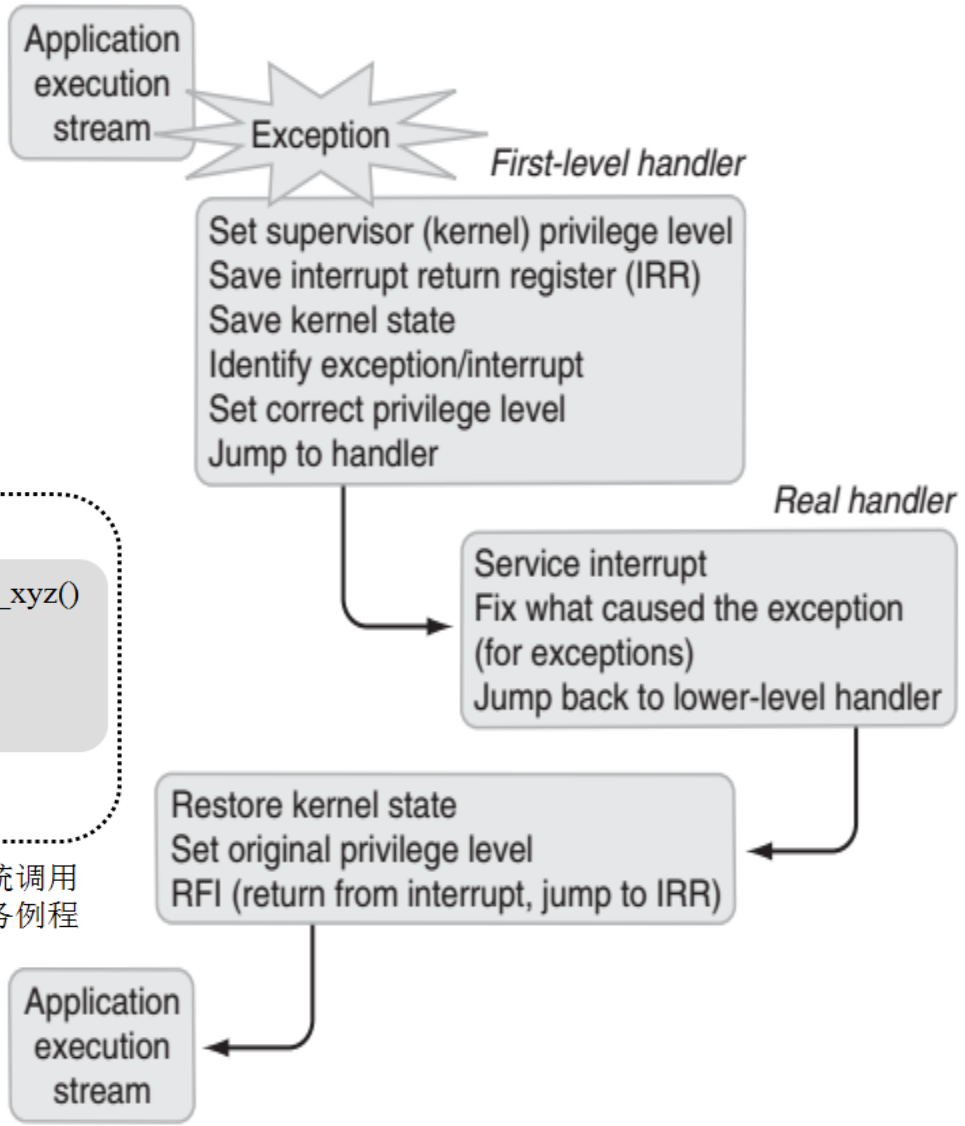
系统调用：应用程序调用OS服务



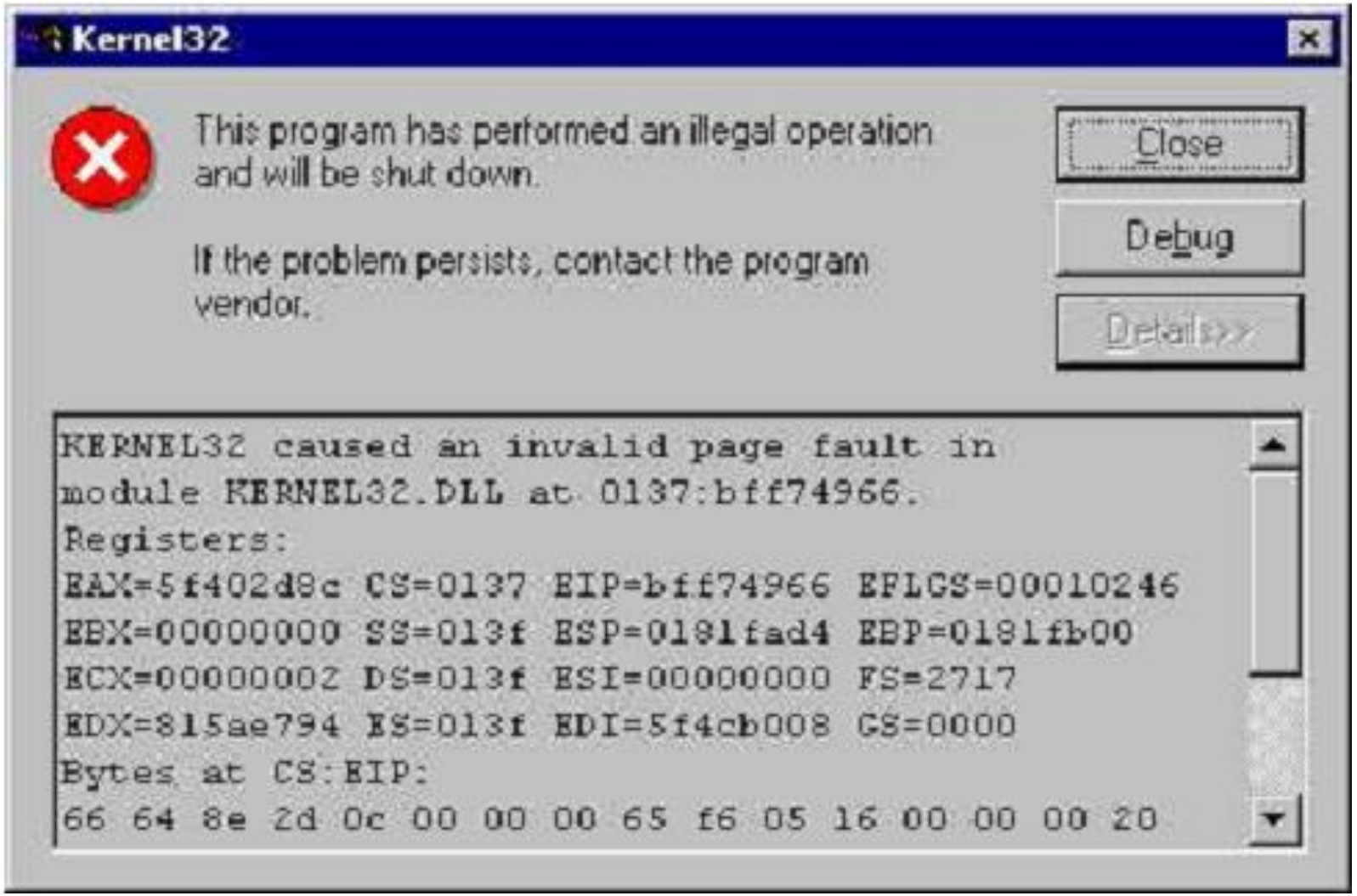
用户态　　内核态



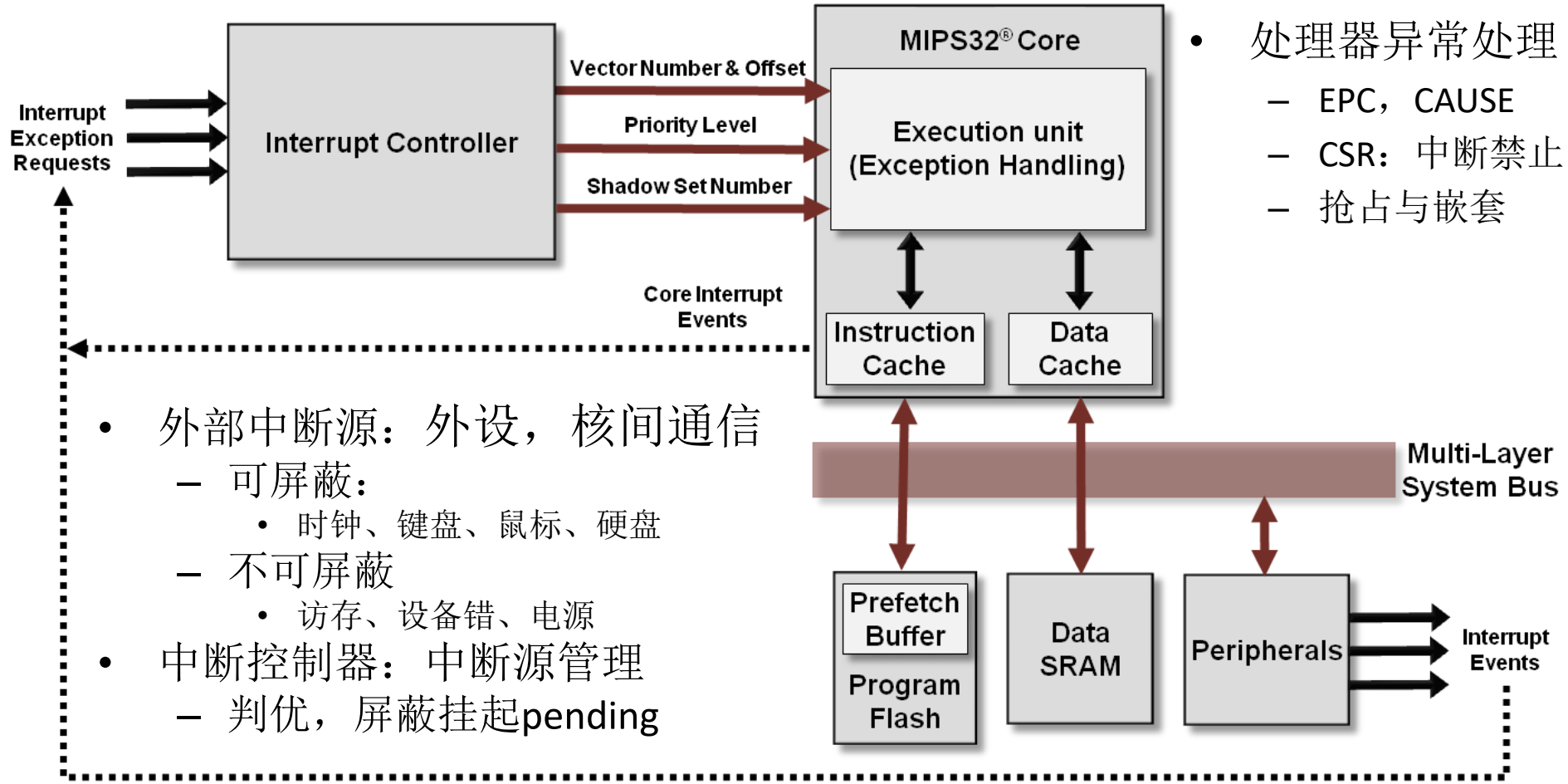
在应用程序调用中的系统调用　在libc标准库中的封装例程　系统调用处理程序　系统调用服务例程



不可恢复异常：异常终止


Kernel32
This program has performed an illegal operation and will be shut down.
If the problem persists, contact the program vendor.
Close
Debug
Details>>
KERNEL32 caused an invalid page fault in
module KERNEL32.DLL at 0137:bff74966.
Registers:
EAX=5f402d8c CS=0137 EIP=bff74966 EFLGS=00010246
EBX=00000000 SS=013f ESP=0181fad4 EBP=0181fb00
ECX=00000002 DS=013f ESI=00000000 FS=2717
EDX=815ae794 ES=013f EDI=5f4cb008 GS=0000
Bytes at CS:EIP:
66 64 8e 2d 0c 00 00 00 65 f6 05 16 00 00 00 20

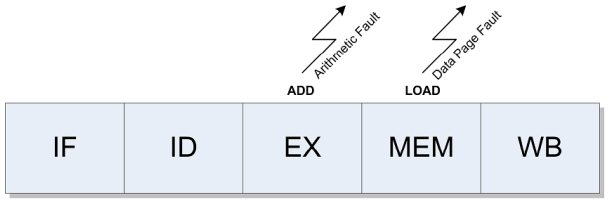
外部中断：中断源，类型，处理机构



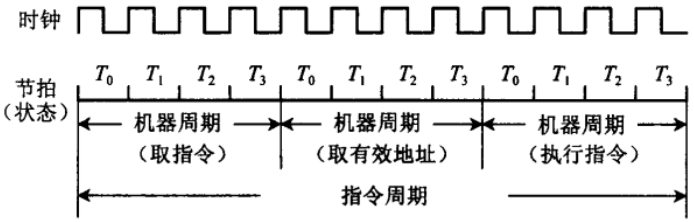
- 处理器异常处理
  - EPC，CAUSE
  - CSR：中断禁止
  - 抢占与嵌套

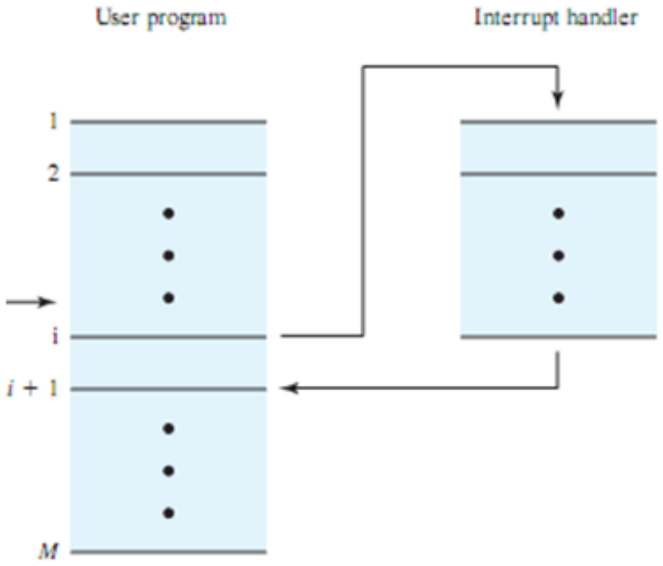
- 外部中断源：外设，核间通信
  - 可屏蔽：
    - 时钟、键盘、鼠标、硬盘
  - 不可屏蔽
    - 访存、设备错、电源
- 中断控制器：中断源管理
  - 判优，屏蔽挂起pending

异常、中断的发生时刻与响应时刻



- 异常：立即响应，同步
  - 指令异常：重启当前指令，或终止执行
  - 系统调用：返回下一条指令
- 中断：指令周期结束响应，异步
  - 返回下一条指令，停止
- 程序序语义：精确断点，p228
  - 之前的指令已执行完成
    - 已经提交其体系结构可见状态
  - 之后的指令还没有发射
    - 没有改变任何机器状态
  - 精确性：顺序/按序，与指令绑定





外部中断发生时刻：异步

- f = (g + h) - (i + j);

```
add x5, x20, x21  // register x5 contains g + h
add x6, x22, x23  // register x6 contains i + j
sub x19, x5, x6   // f gets x5 – x6, which is (g + h) – (i + j)
```

- A[12] = h + A[8];

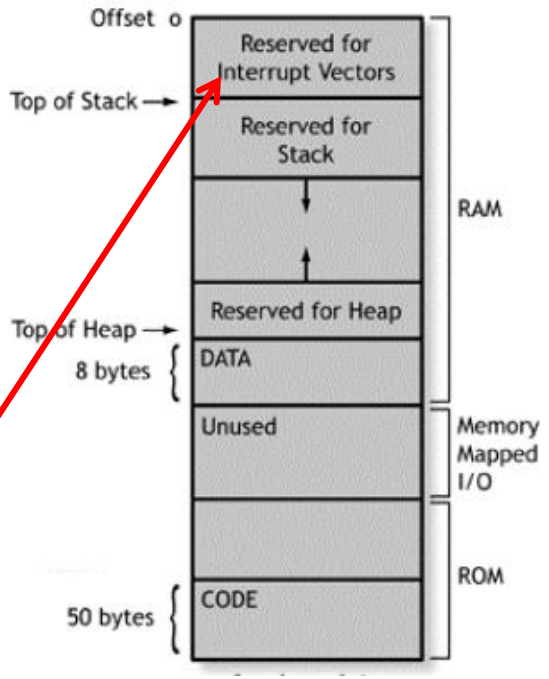
  ld x9, 64(x22) // Temporary reg x9 gets A[8]

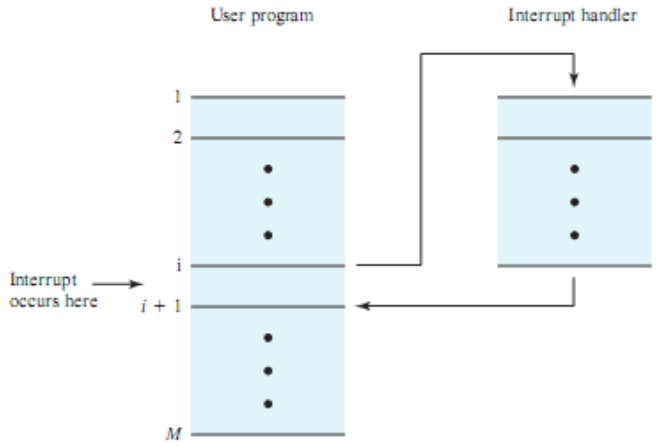
  add x9, x21, x9 // Temporary reg x9 gets h + A[8]

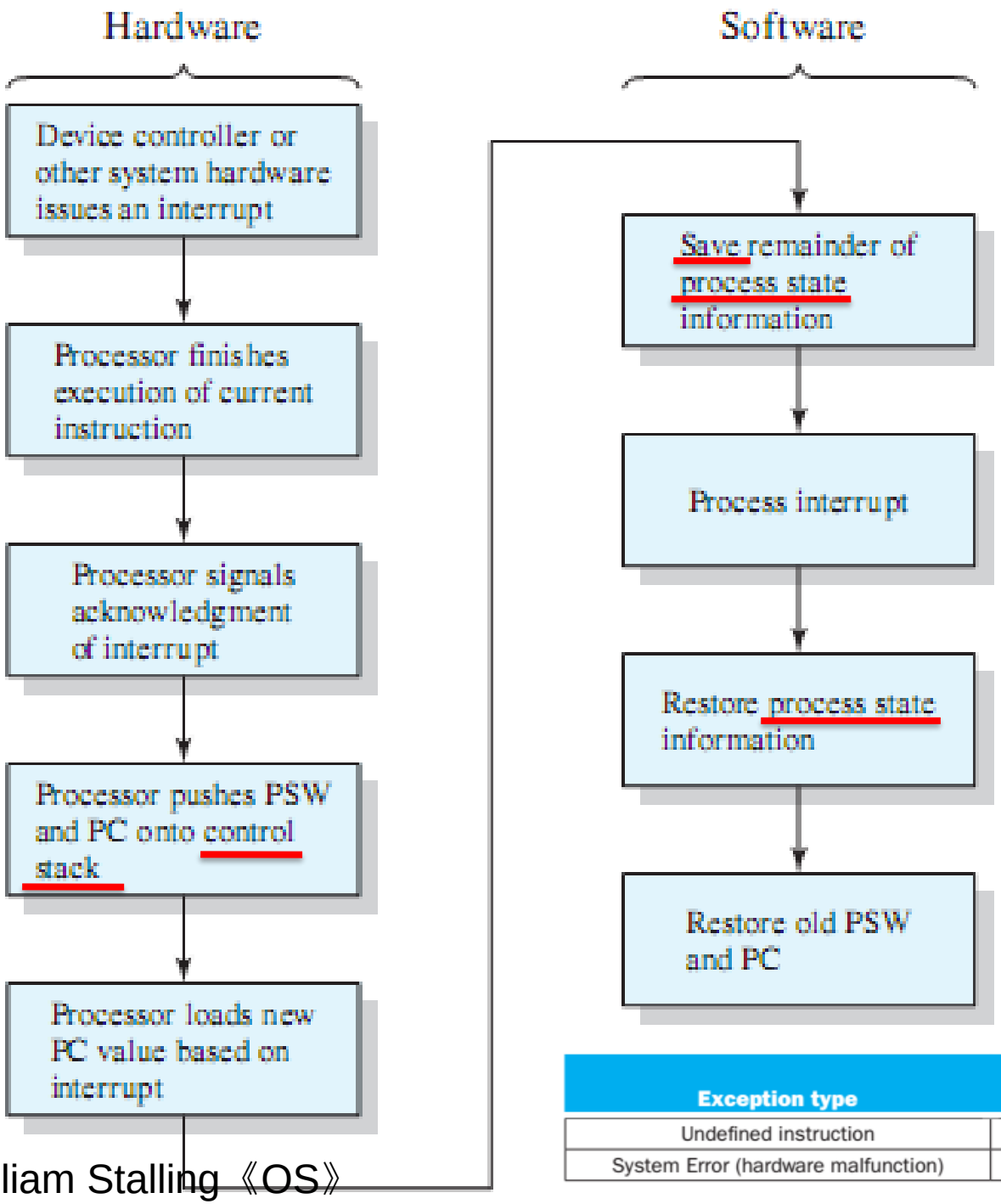
  sd x9, 96(x22) // Stores h + A[8] back into A[12]

The Basics of Exception Handling

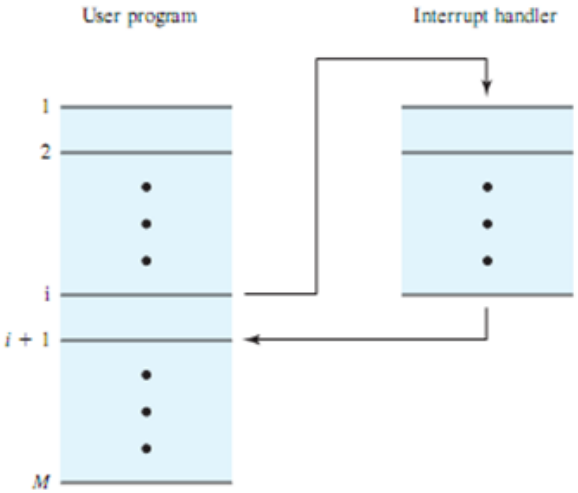
- 软硬件接口约定（p228）
  - 硬件
    - EPC：断点=异常指令地址/下一条指令地址
    - Cause：检测并记录异常原因
    - 关中断，清当前中断
    - 保存PSW/CSR
    - *建立处理环境：**user mode => kernel mode***
    - 转ESR/ISR入口：非向量式/向量式（Vector表）
  - 软件
    1. 保存剩余现场：RF
    2. *开中断*
    3. 进行异常服务
    4. 关中断
    5. 恢复现场：RF
    6. 中断返回：iret/eret
       - 开中断，*恢复用户模式*，恢复PSW







中断处理：软硬件接口约定

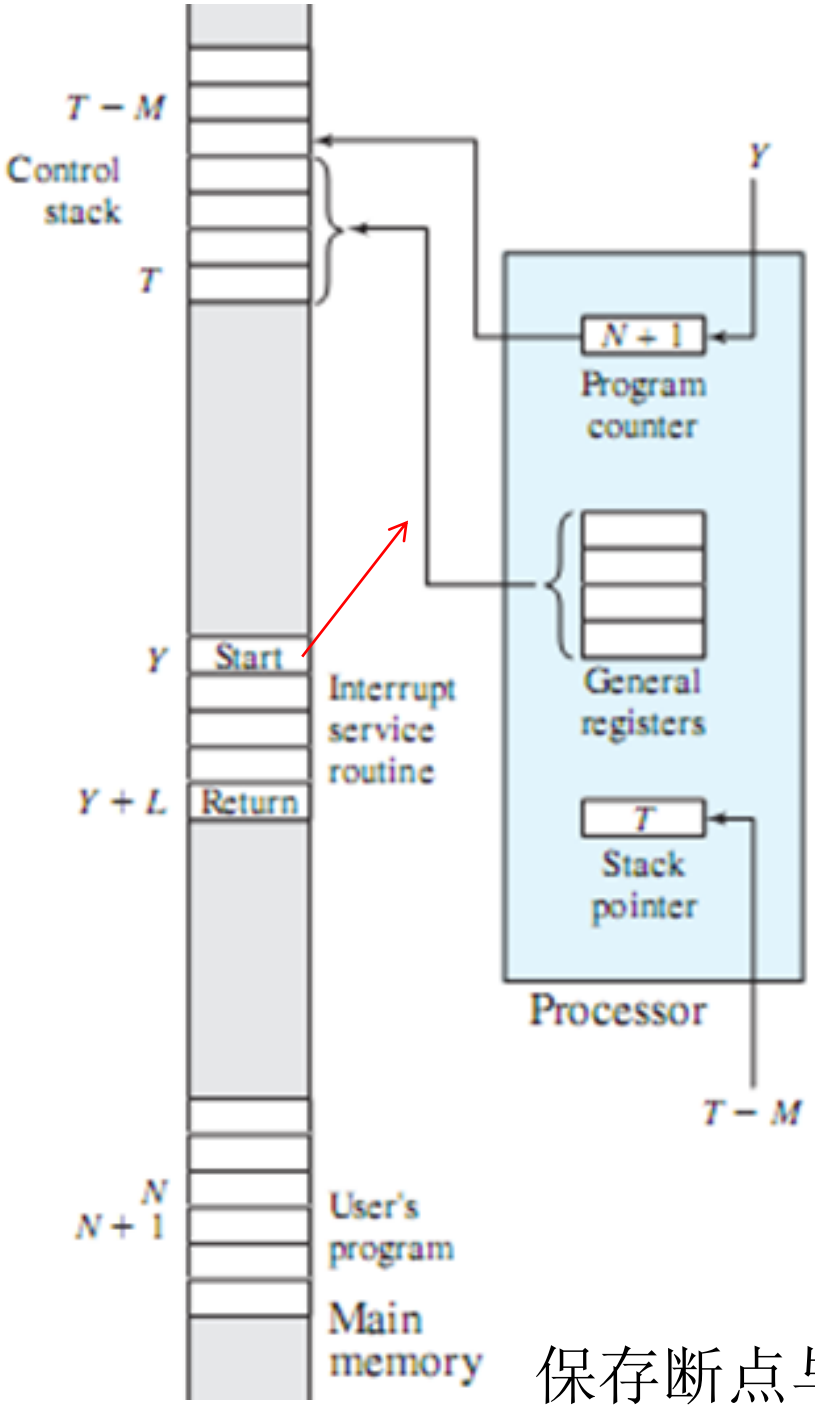


ESR/ISR入口（向量式）RV$4.9.1

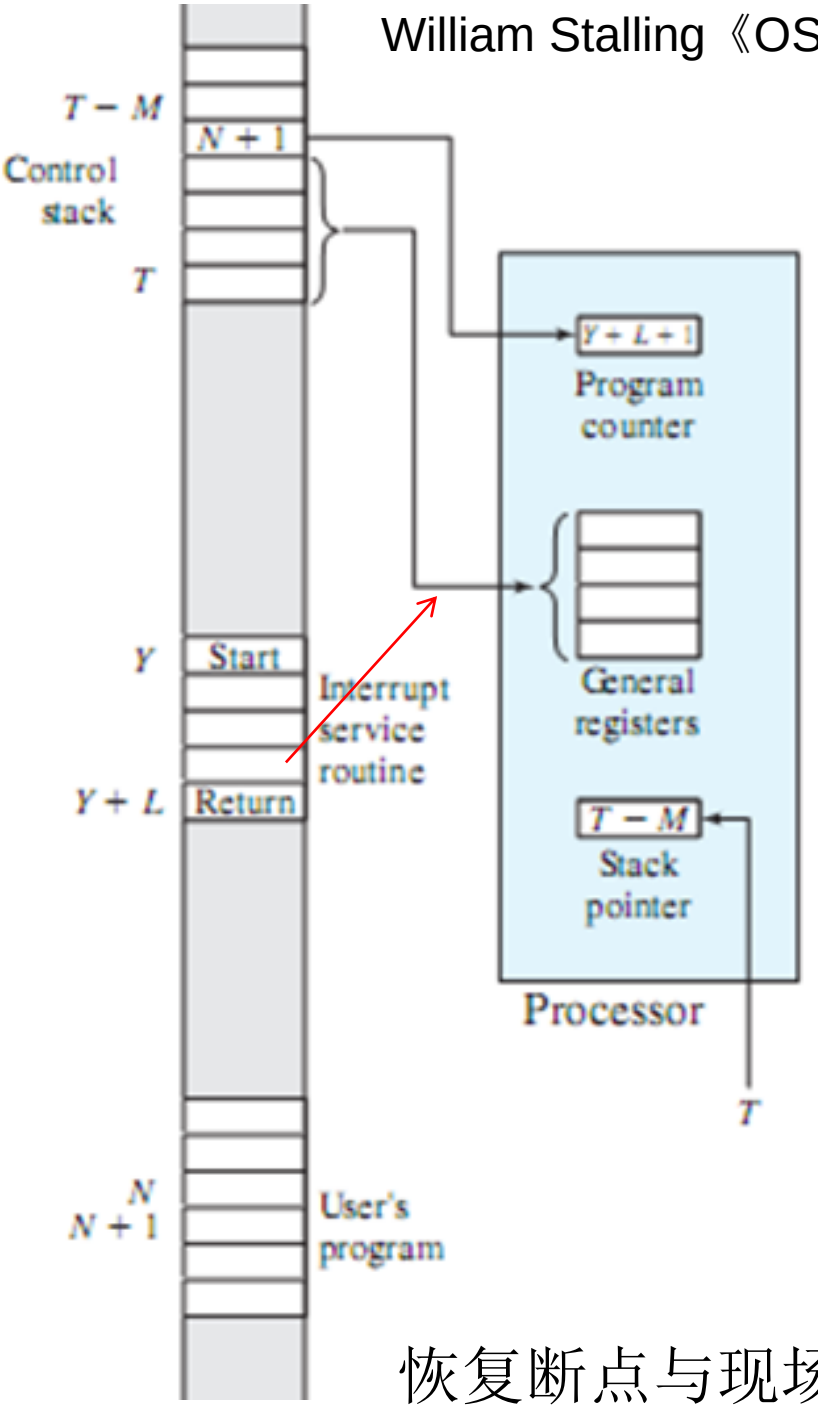
Exception type	Exception vector address to be added to a Vector Table Base Register
Undefined instruction	$00\ 0100\ 0000_{two}$
System Error (hardware malfunction)	$01\ 1000\ 0000_{two}$

William Stalling《OS》

William Stalling《OS》



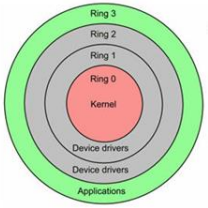
保存断点与现场



恢复断点与现场

RV Control and Status Register

- RV工作模式：user mode，supervisor mode，machine mode
- CSR：4K独立的MMIO地址空间
  - 不同工作模式各有一套CSR
    - 公共：Status，EPC，Cause，*Scratch（用于模式切换）。。。*
  - 读写模式：*read-modify-write* cycle

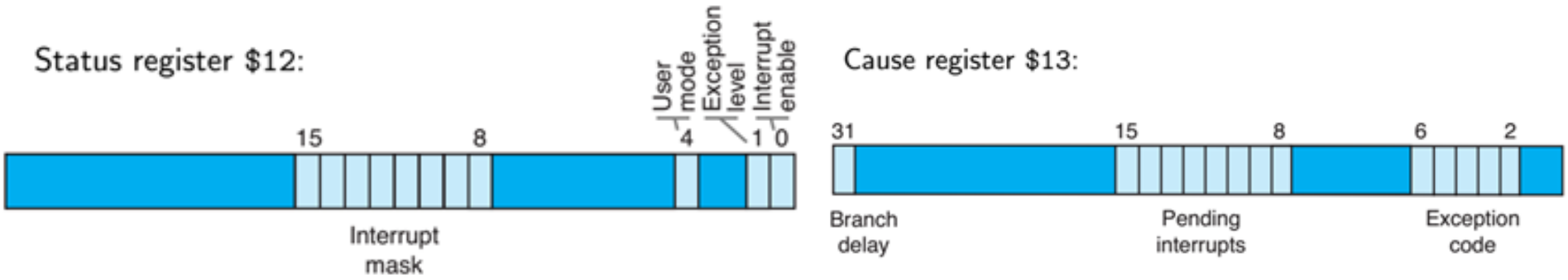


Number	Privilege	Name	Description
		User Trap Setup	
0x000	URW	ustatus	User status register.
0x004	URW	uie	User interrupt-enable register.
0x005	URW	utvec	User trap handler base address.
		User Trap Handling	
0x040	URW	uscratch	Scratch register for user trap handlers.
0x041	URW	uepc	User exception program counter.
0x042	URW	ucause	User trap cause.
0x043	URW	ubadaddr	User bad address.
0x044	URW	uip	User interrupt pending.

User mode CSR

The RISC-V Instruction Set Manual Volume I: Unprivileged ISA
The RISC-V Instruction Set Manual Volume II: Privileged Architecture

Important Exceptions registers： MIPS



Number	Name	Cause of exception
0	Int	interrupt (hardware)
4	AdEL	address error exception (load or instruction fetch)
5	AdES	address error exception (store)
6	IBE	bus error on instruction fetch
7	DBE	bus error on data load or store
8	Sys	syscall exception
9	Bp	breakpoint exception
10	RI	reserved instruction exception
11	CpU	coprocessor unimplemented
12	Ov	arithmetic overflow exception
13	Tr	trap
15	FPE	floating point

访问CSR，$5.14

- CSR swap instructions
  - 6条：I-type，12位=4K
  - 将CSR复制到通用Reg，用寄存器操作数或Imm改写它们。

RV图5-47（Fig 5-48）

Type	Mnemonic	Name
CSR Access	CSRRWI	CSR Read/Write Immediate
	CSRRSI	CSR Read/Set Immediate
	CSRRCI	CSR Read/Clear Immediate
	CSRRW	CSR Read/Write
	CSRRS	CSR Read/Set
	CSRRC	CSR Read/Clear

《The RISC-V Instruction Set Manual》

31 20	19 15	14 12	11 7	6 0
csr	rs1	funct3	rd	opcode
12	5	3	5	7
source/dest	source	CSRRW	dest	SYSTEM
source/dest	source	CSRRS	dest	SYSTEM
source/dest	source	CSRRC	dest	SYSTEM
source/dest	uimm[4:0]	CSRRWI	dest	SYSTEM
source/dest	uimm[4:0]	CSRRSI	dest	SYSTEM
source/dest	uimm[4:0]	CSRRCI	dest	SYSTEM

ISR编程： RV Arch'ed Regs

Name	Register number	Usage	Preserved on call?
x0	0	The constant value 0	n.a.
x1 (ra)	1	Return address (link register)	yes
x2 (sp)	2	Stack pointer	yes
x3 (gp)	3	Global pointer	yes
x4 (tp)	4	Thread pointer	yes
x5-x7	5–7	Temporaries	no
x8-x9	8–9	Saved	yes
x10-x17	10–17	Arguments/results	no
x18-x27	18–27	Saved	yes
x28-x31	28–31	Temporaries	no

RV 图2-14

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller

Table 25.1

The RISC-V Instruction Set Manual Volume I

过程调用时可用a0~a7，s0~s11，t0~t6

ABI calling convention：——减少代码量

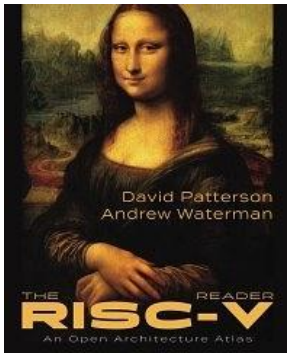
- **a/t**-registers are caller-saved
- **s**-regs are callee-saved and **preserve** their contents across function calls

TimerISR

非向量式中断

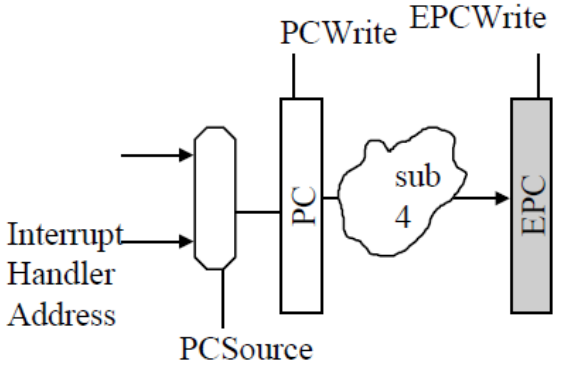
```
# save registers
csrrw a0, mscratch, a0   # save a0; set a0 = &temp storage
sw a1, 0(a0)             # save a1
sw a2, 4(a0)             # save a2
sw a3, 8(a0)             # save a3
sw a4, 12(a0)            # save a4

# decode interrupt cause
csrr a1, mcause          # read exception cause
bgez a1, exception       # branch if not an interrupt
andi a1, a1, 0x3f        # isolate interrupt cause
li a2, 7                 # a2 = timer interrupt cause
bne a1, a2, otherInt     # branch if not a timer interrupt

# handle timer interrupt by incrementing time comparator
la a1, mtimecmp          # a1 = &time comparator
lw a2, 0(a1)             # load lower 32 bits of comparator
lw a3, 4(a1)             # load upper 32 bits of comparator
addi a4, a2, 1000        # increment lower bits by 1000 cycles
sltu a2, a4, a2          # generate carry-out
add a3, a3, a2           # increment upper bits
sw a3, 4(a1)             # store upper 32 bits
sw a4, 0(a1)             # store lower 32 bits

# restore registers and return
lw a4, 12(a0)            # restore a4
lw a3, 4(a0)             # restore a3
lw a2, 4(a0)             # restore a2
lw a1, 0(a0)             # restore a1
csrrw a0, mscratch, a0   # restore a0; mscratch = &temp storage
mret                     # return from handler
```

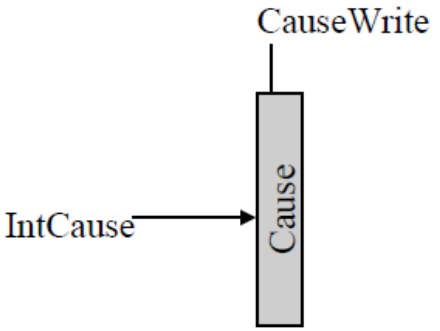
图10.6



处理器指令异常实现：四步法

- 异常处理过程分解、定义微操作
  - 保存断点：PC-4 => EPC
  - 保存异常原因：Cause寄存器
    - 非法指令：在ID周期
    - 算术溢出：在EXE周期
    - *缺页*
  - 转ESR：入口00 0010 0000，非向量式
  - *ESR@OS*
    - *根据cause分别处理*
      - *非法指令：为用户程序提供某些服务*
      - *溢出：报错*
    - *返回：ERET (return-from-environment)*
- 数据通路、控制信号、综合FSM
  - EPC，Cause，转Vec
    - IntCause：0-非法指令，1-溢出
  - EPCWrite，CauseWrite，分支



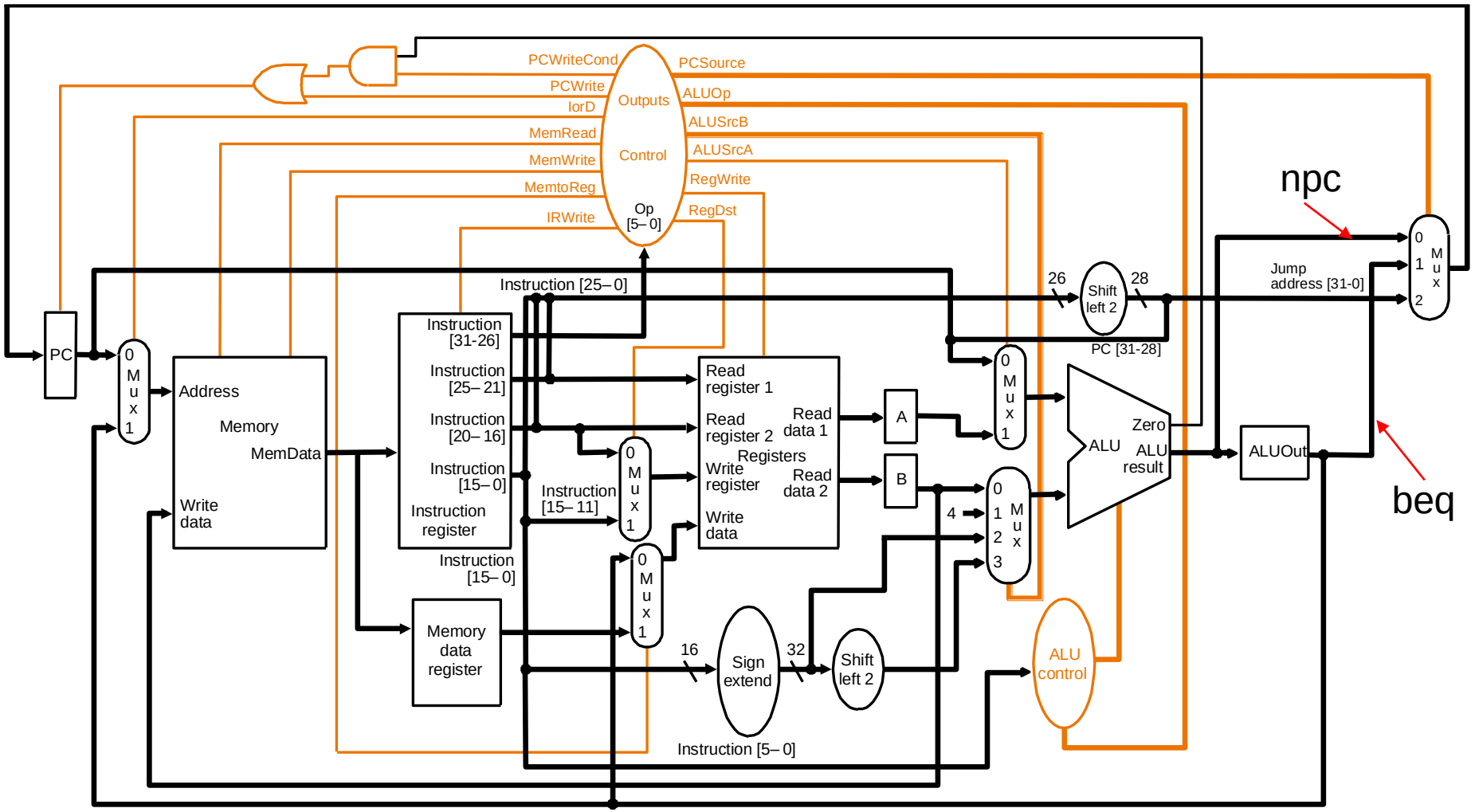


RV$4.9.1 ESR/ISR入口地址

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

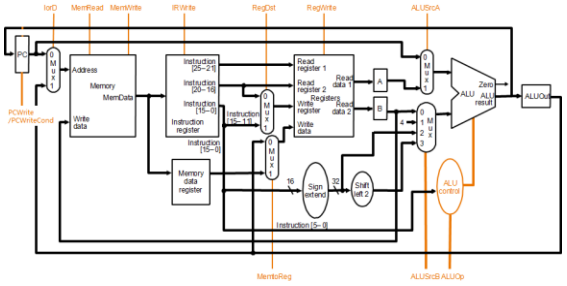
多周期总图

IF	ID	EX	MEM	WB



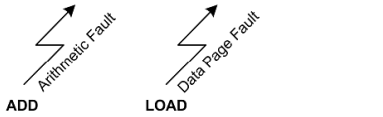
多周期控制器FSM

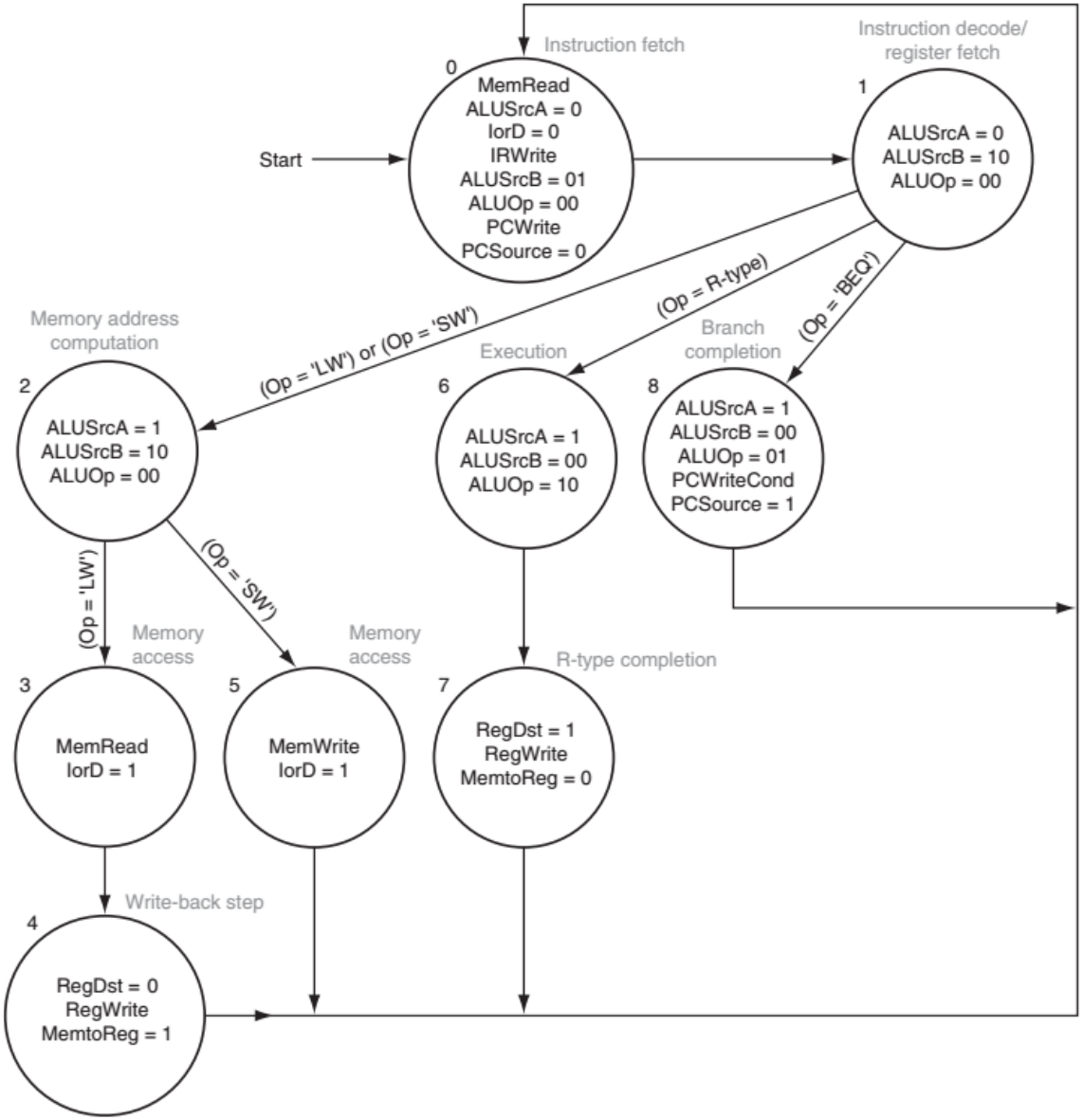
每个状态一个时钟周期。



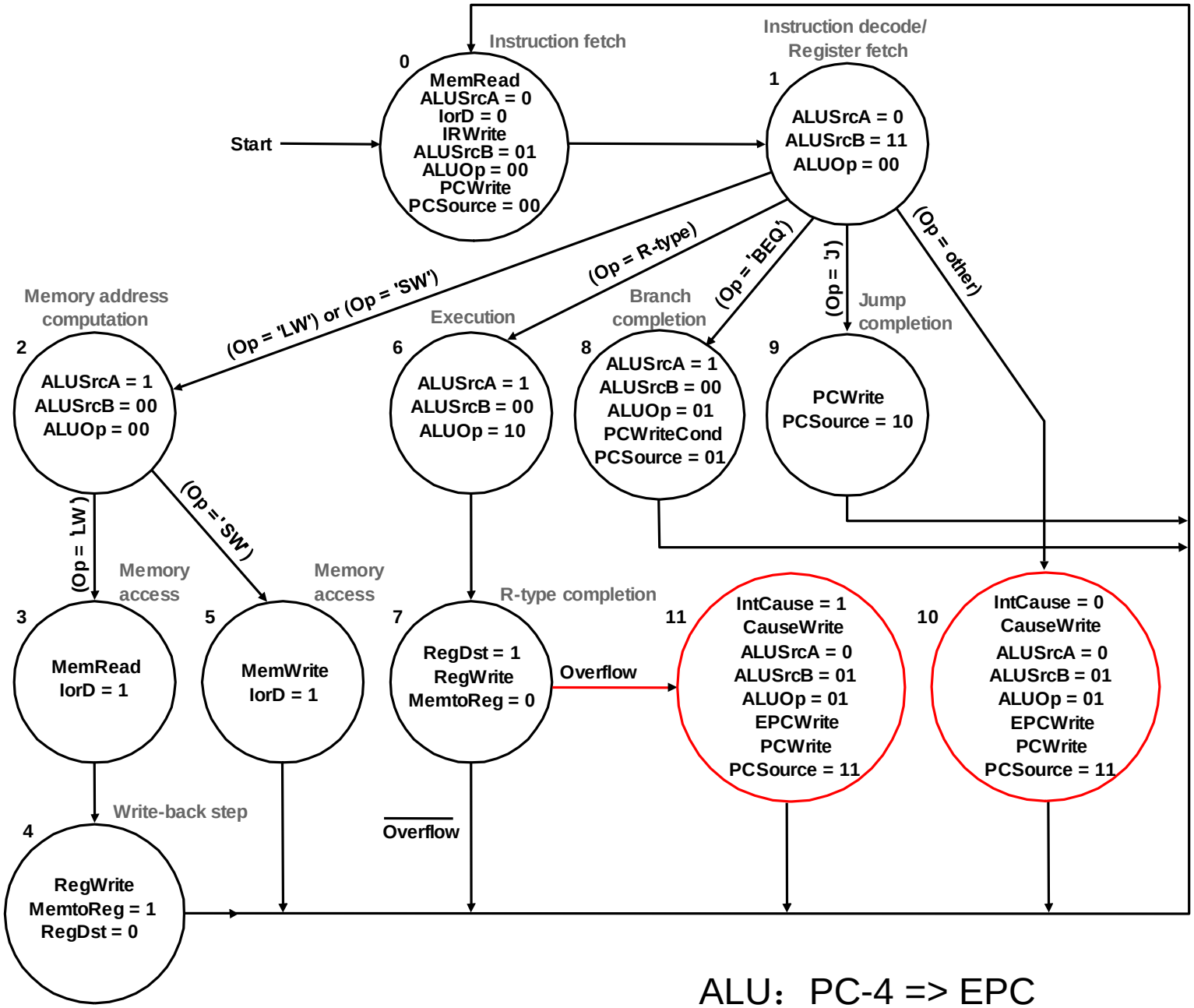
指令异常：

- 非法指令
- 算术溢出
- *缺页*



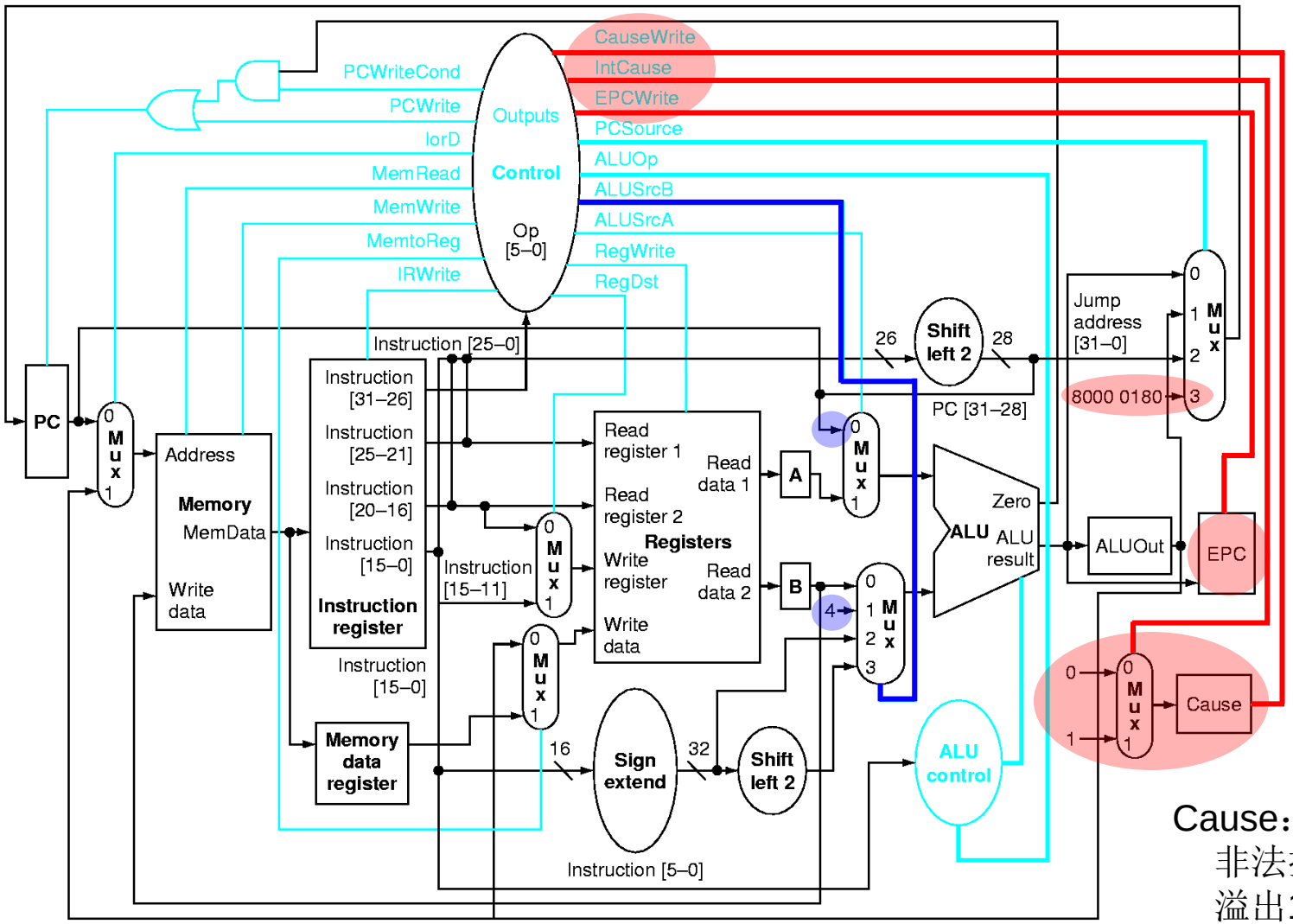


异常处理控制



ALU：PC-4 => EPC

Exceptions Handling in Multi-Cycle MIPS

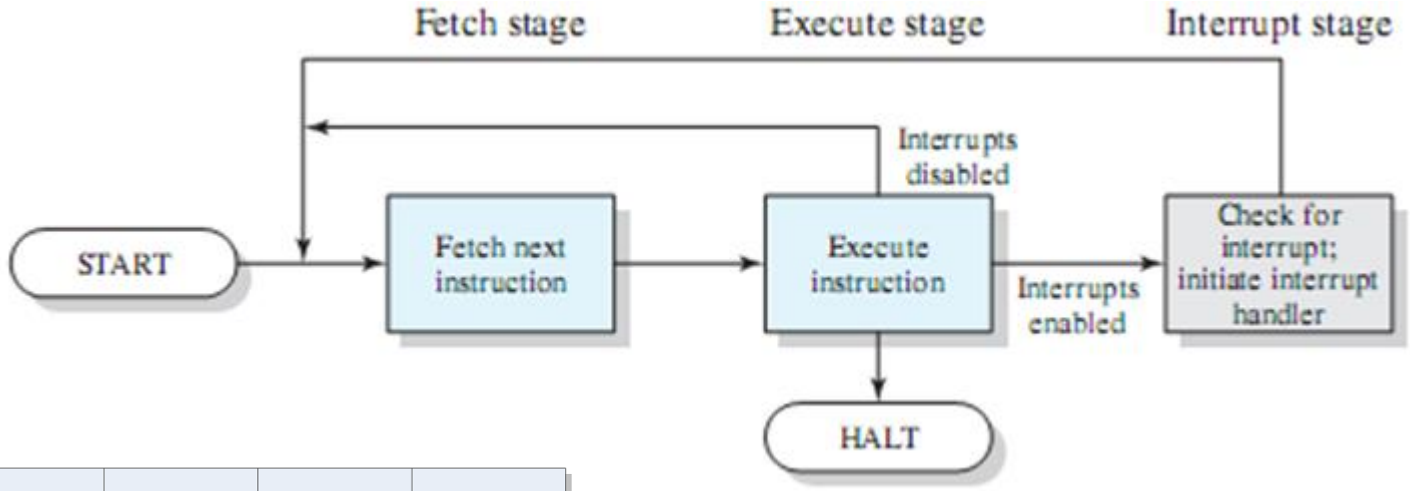


Cause：

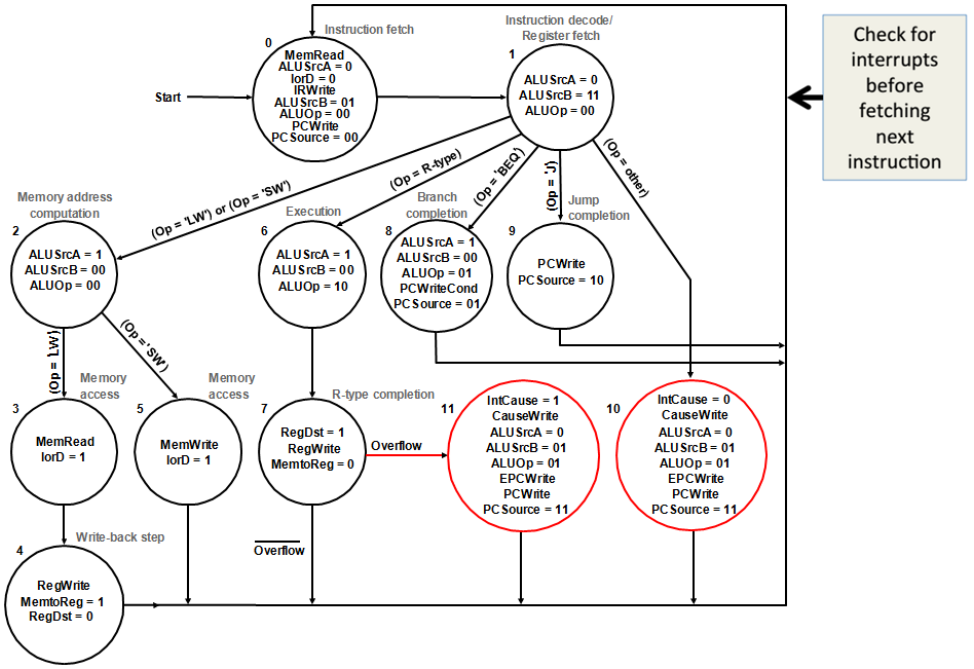
非法指令0

溢出1

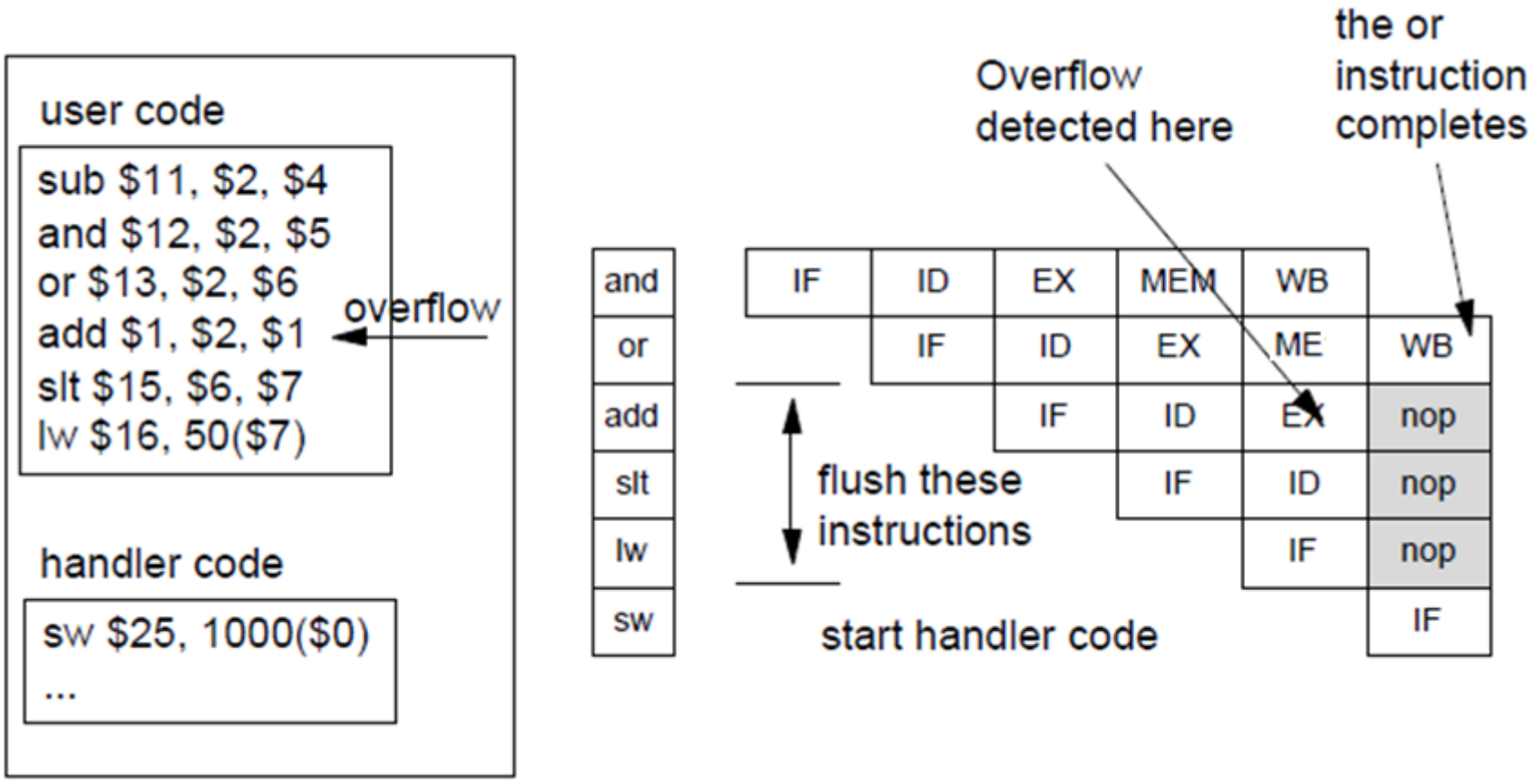
Multicycle Interrupts：中断周期



- CPU
  - Enable
  - EPC，Cause
- IntrCtrler
  - Mask，判优



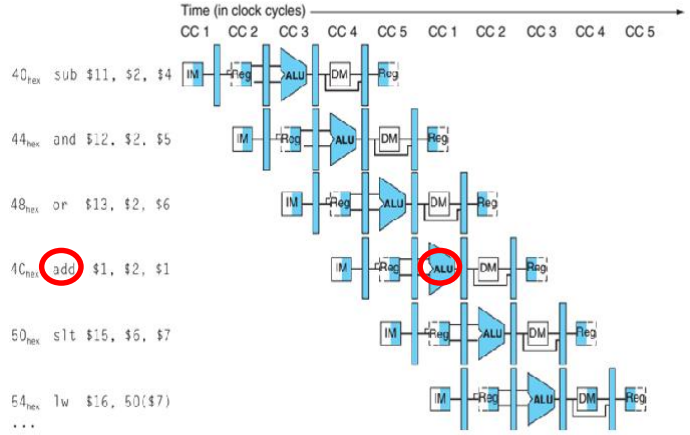
$4.9.2：add指令溢出异常的响应



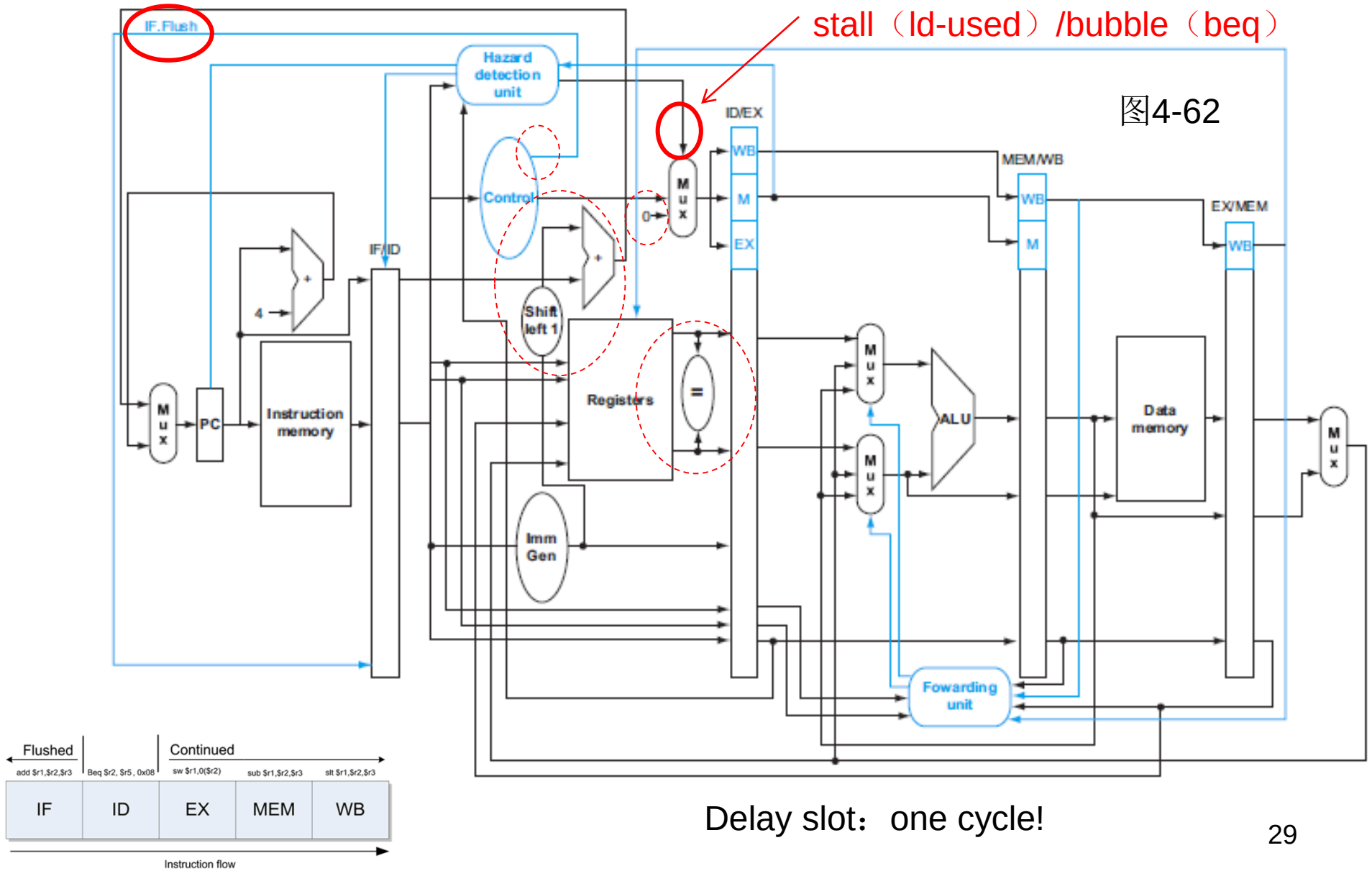
- 保证异常响应的顺序与指令执行的逻辑顺序相同
  - 顺序语义：之前的指令完成，之后的指令取消

RV的add指令溢出异常处理$4.9.2

- 将异常视为一种控制相关
  - 执行完之前的所有指令
  - flush异常指令及之后的所有指令
    - add：IF.Flush，ID.Flush，EX.Flush
  - 记录异常原因：Cause
  - 保存断点：EPC = 异常指令PC，精确！
  - 转异常处理程序：非向量中断（RV tvec@CSR）
  - 异常返回：EPC，重新执行异常指令
    - “溢出”一般放弃，其他（缺页）可能继续
- 遵从“简约原则”
  - 硬件仅提供最少支持（EPC、Cause），其他由OS负责（如分析Cause）

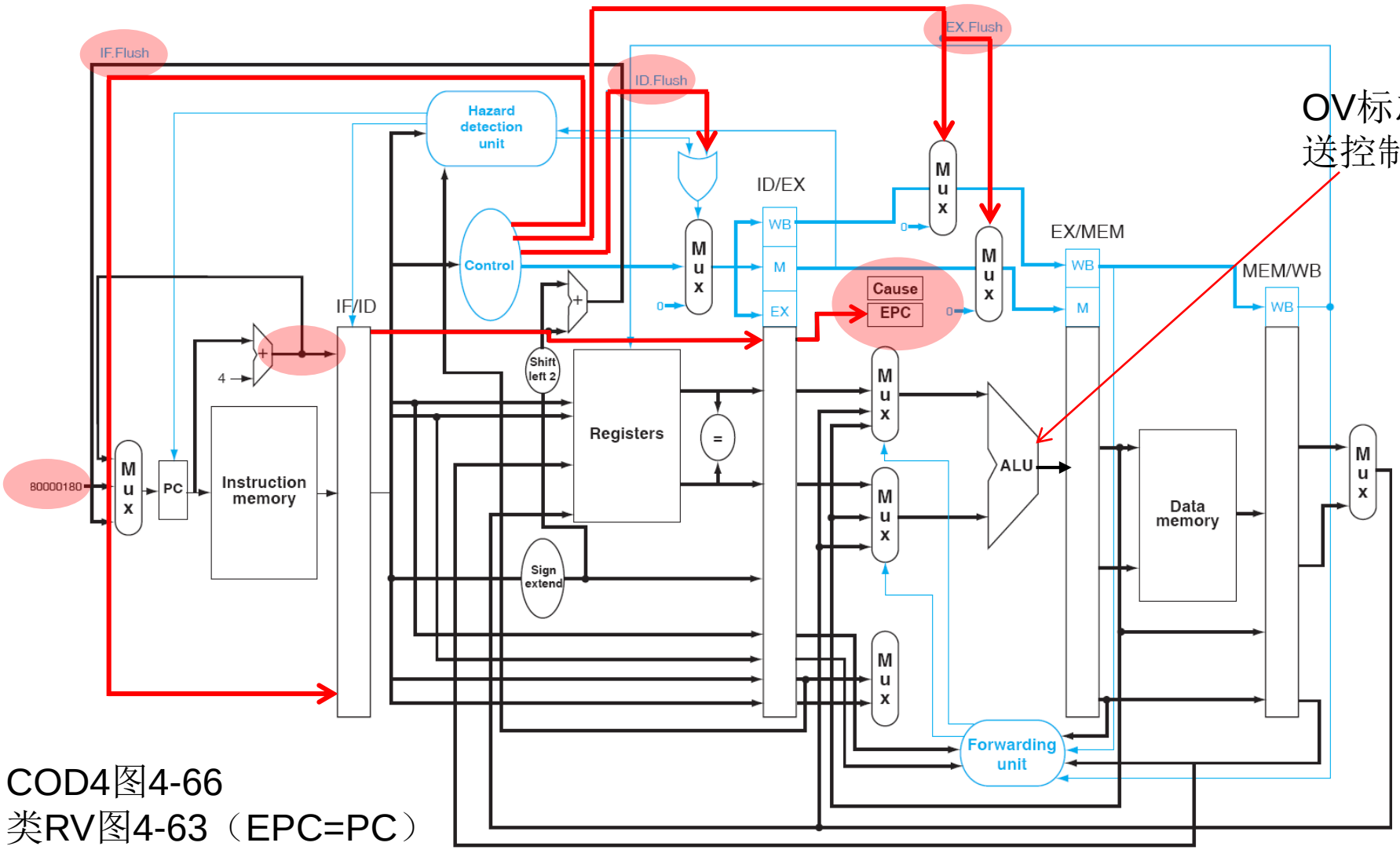


单周期beq实现：IF.Flush，bubble



Delay slot：one cycle!

add指令overflow： flush IF/ID/EX



OV标志送控制器

COD4图4-66

类RV图4-63（EPC=PC）

当前周期：检测，记录（EPC、CAUSE），排空，ESR入口写nPC

下一个周期：取ESR的第一条指令

RV图4-64，CC6

```
40hex   sub    x11, x2, x4
44hex   and    x12, x2, x5
48hex   or     x13, x2, x6
4Chex   add    x1,  x2, x1
50hex   sub    x15, x6, x7
54hex   ld     x16, 100(x7)
. . .
```

Overflow detected deasserting add flush EX

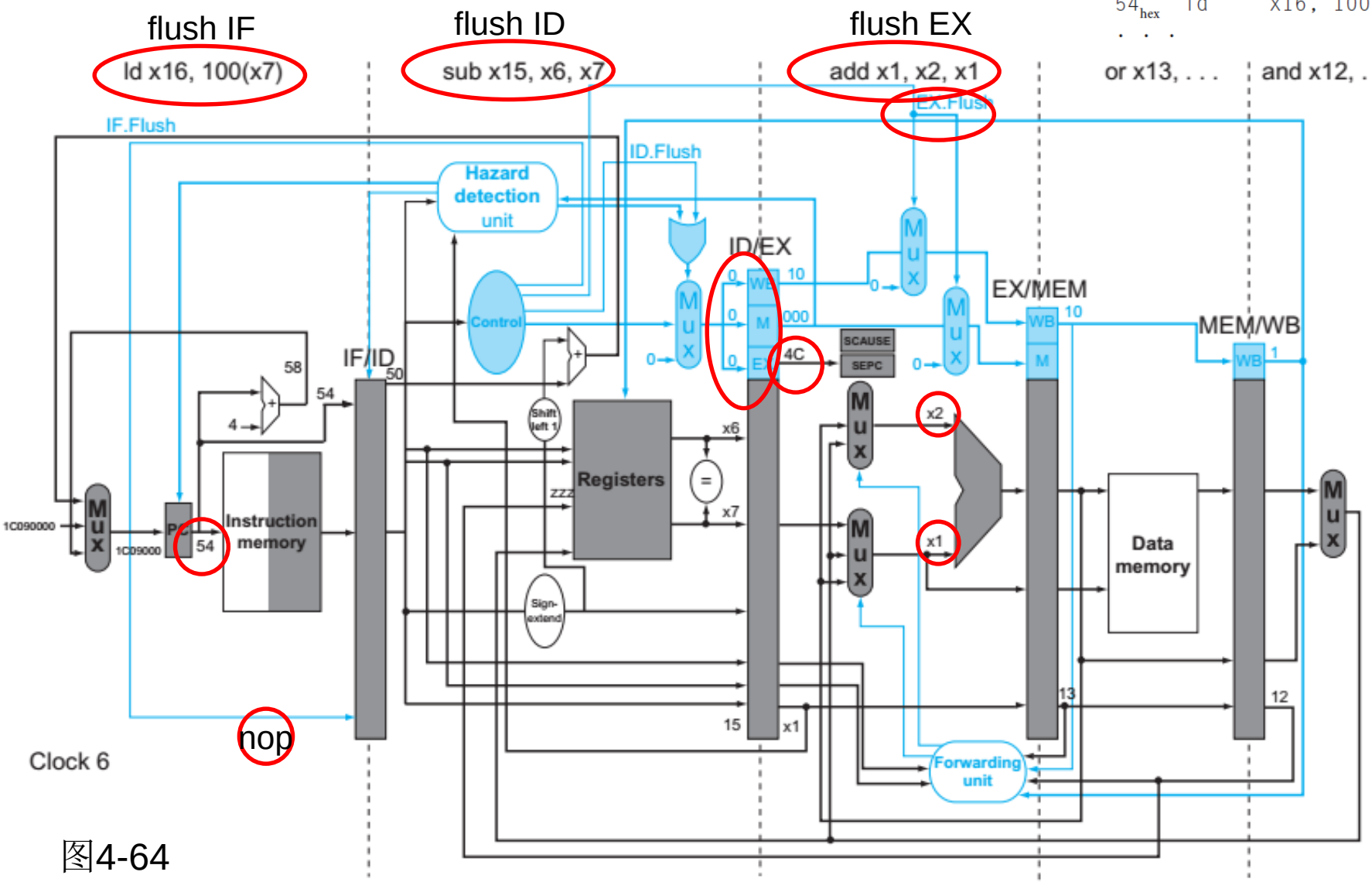


图4-64

本例EPC=4C，精确；如果EPC=58，非精确

RV图4-64，CC7

```
40hex   sub    x11, x2, x4
44hex   and    x12, x2, x5
48hex   or     x13, x2, x6
4Chex   add    x1,  x2, x1
50hex   sub    x15, x6, x7
54hex   ld     x16, 100(x7)
. . .
```

first instruction of exception routine

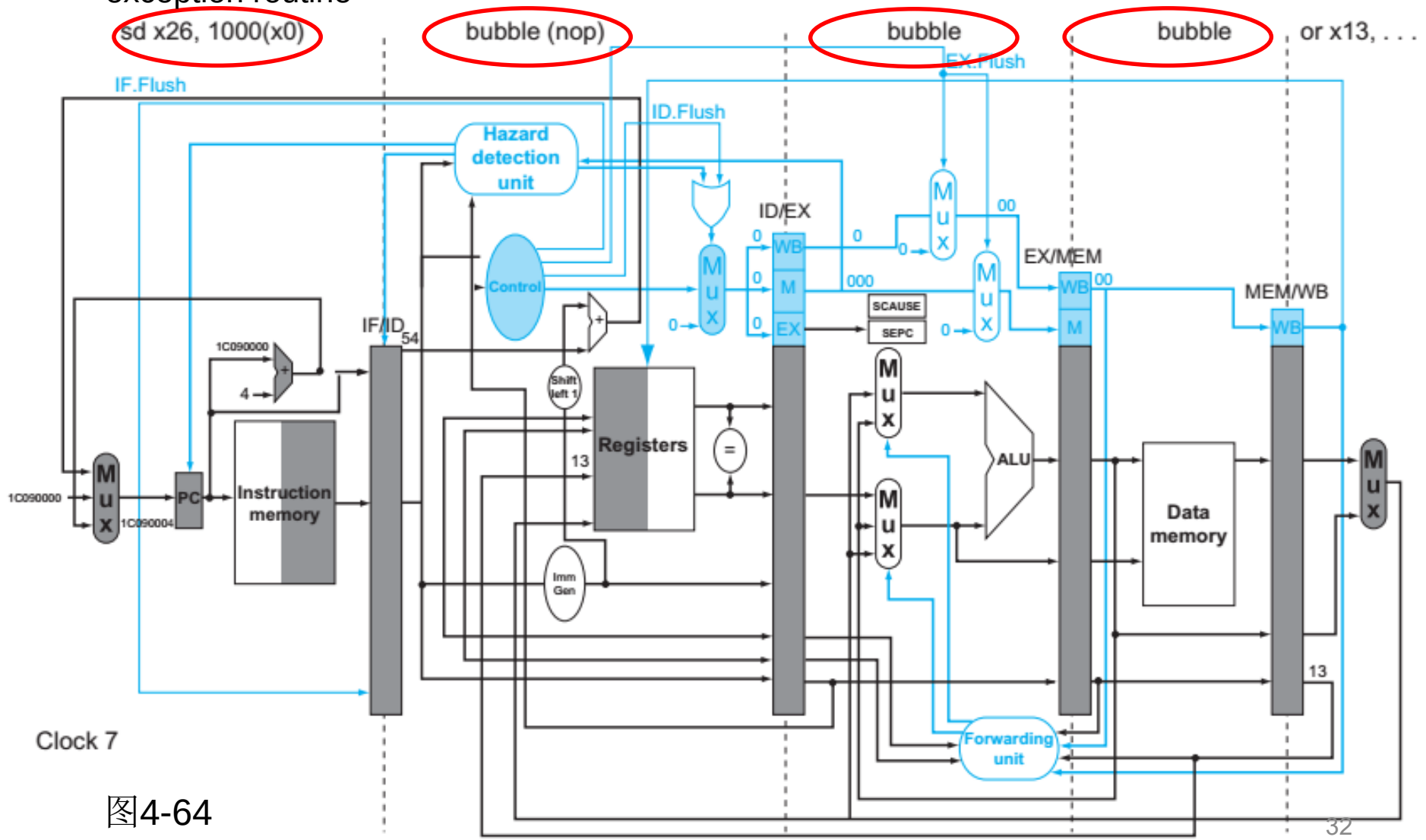
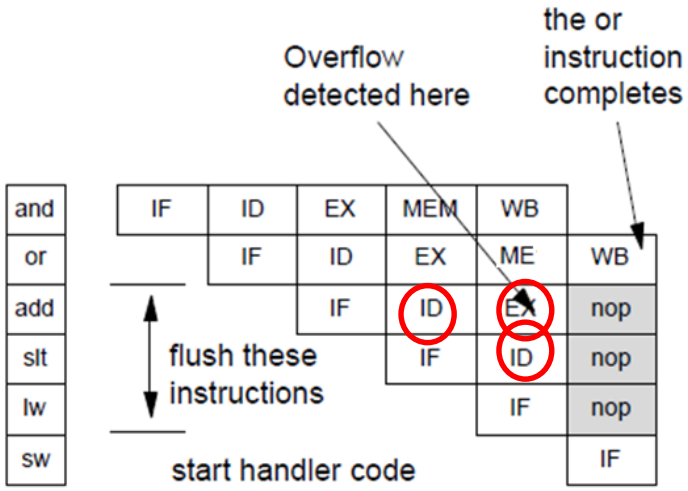


图4-64

上例的异常处理的问题：仅单指令EX段异常！

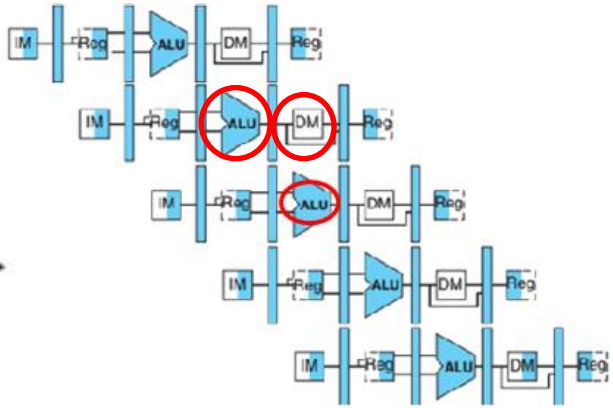
- 单指令多异常： FCFS
  - IF段：缺页
  - ID段：非法指令
  - EX段：溢出
  - MEM段：缺页
  - WB段：无（写寄存器错？）
  - 各段的异常在本段处理？
    - 复杂☹：EPC，Cause，Flush

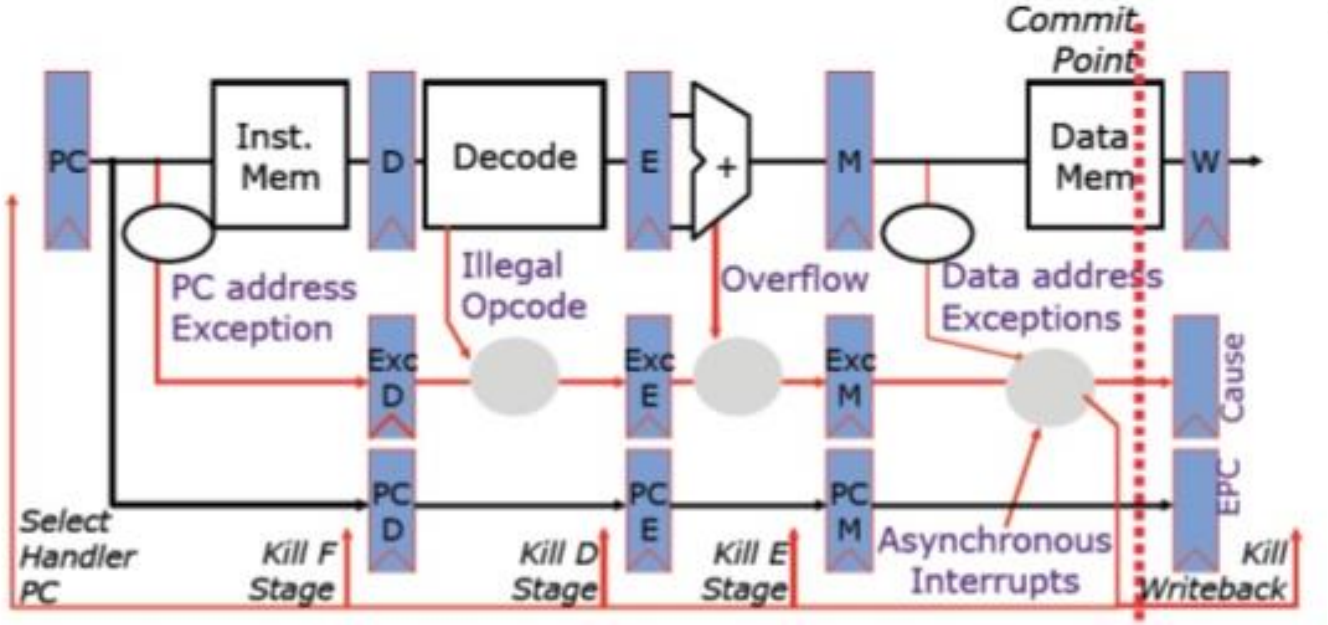


- 多指令同一周期异常： FCFS
  - 精确（RV）：流水线中的异常与引发异常指令精确关联
    - EPC=异常指令，如add
  - 非精确（DLX）：异常与执行不精确关联，如EPC=IF段指令
    - 硬件简单，但软件复杂：需要OS依据流水线深度确定EPC
- 异常嵌套？

多异常：精确异常实现技术

- 指令异常提交点：哪一段？
  - 并不立即响应，仅标记（并气泡？），直至提交点
  - 多指令：响应流水线中最早的异常，kill新的异常
- 发生：Flush&重定向PC





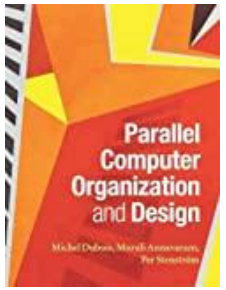
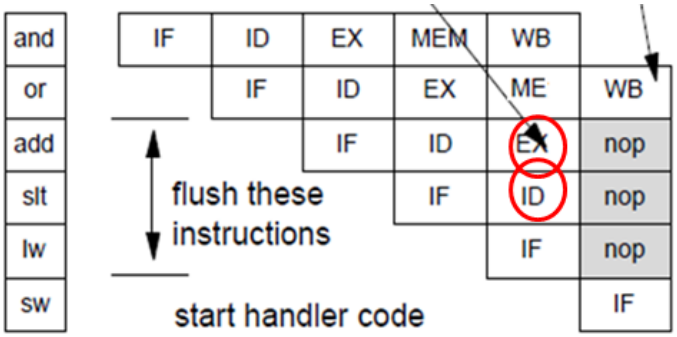
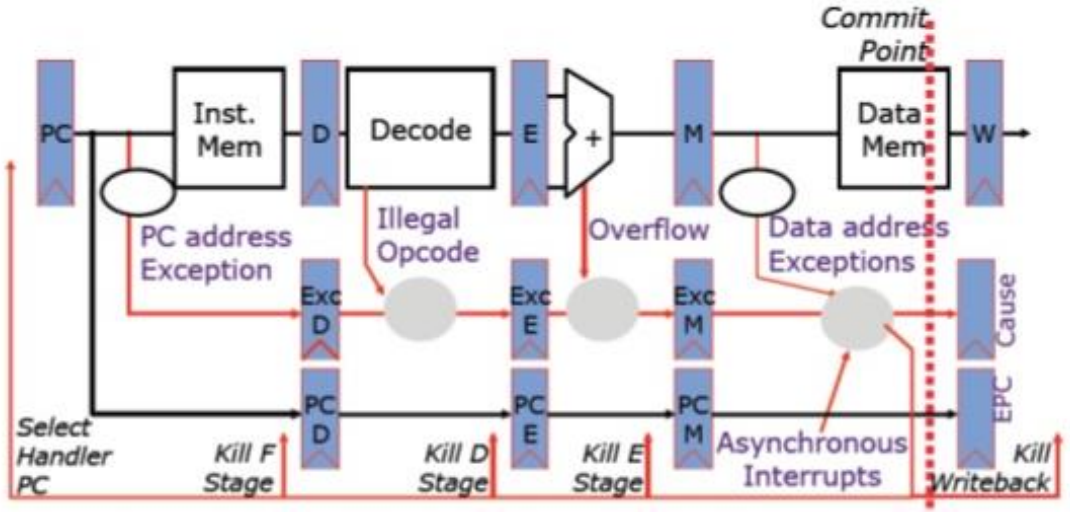


Fig11.3 $3.3.1

流水线中的中断响应？

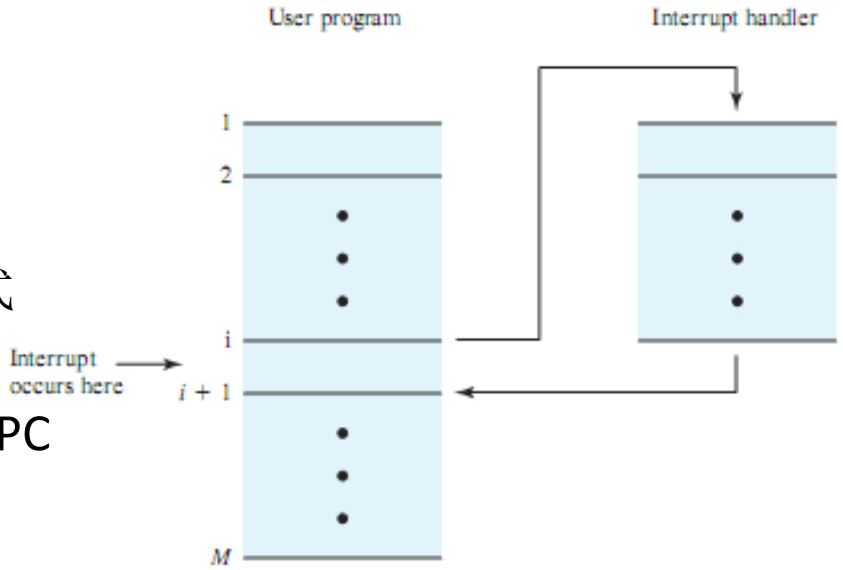
- 中断的非精确性：不要求“精确”
  - p227：“I/O设备请求和硬件故障与指令无关，因此中断发生时何时中断流水线具有较大的灵活性”
  - 中断响应：停止IF取指（EPC），完成已进入流水线的指令，转ISR
    - 出现异常？
- 提交点：外部中断注入点：按优先级
  - 中断：EPC = 注入点IF段的当前指令
  - 多中断：优先级，嵌套？
  - 异常+中断：优先级？



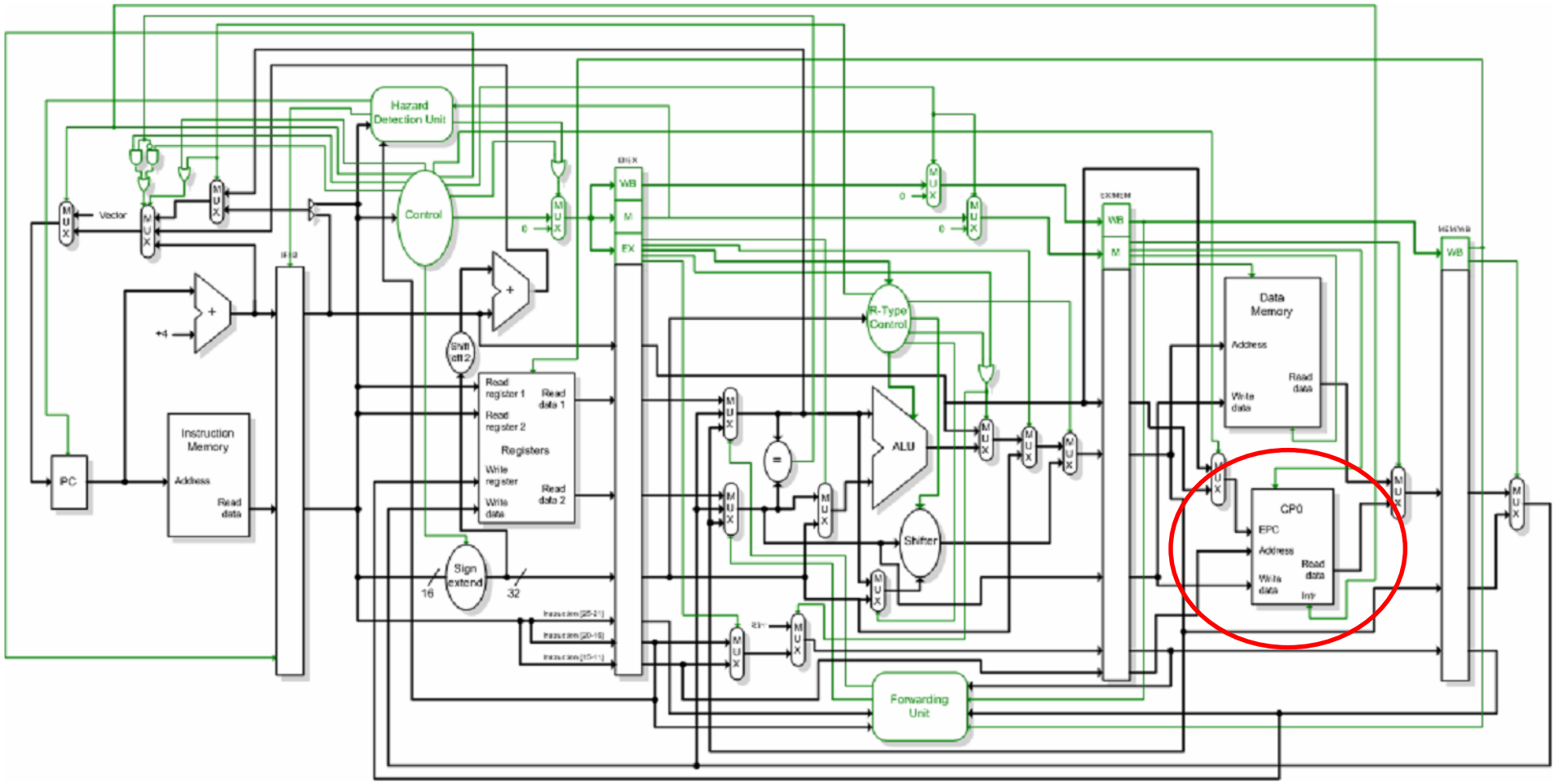


小结

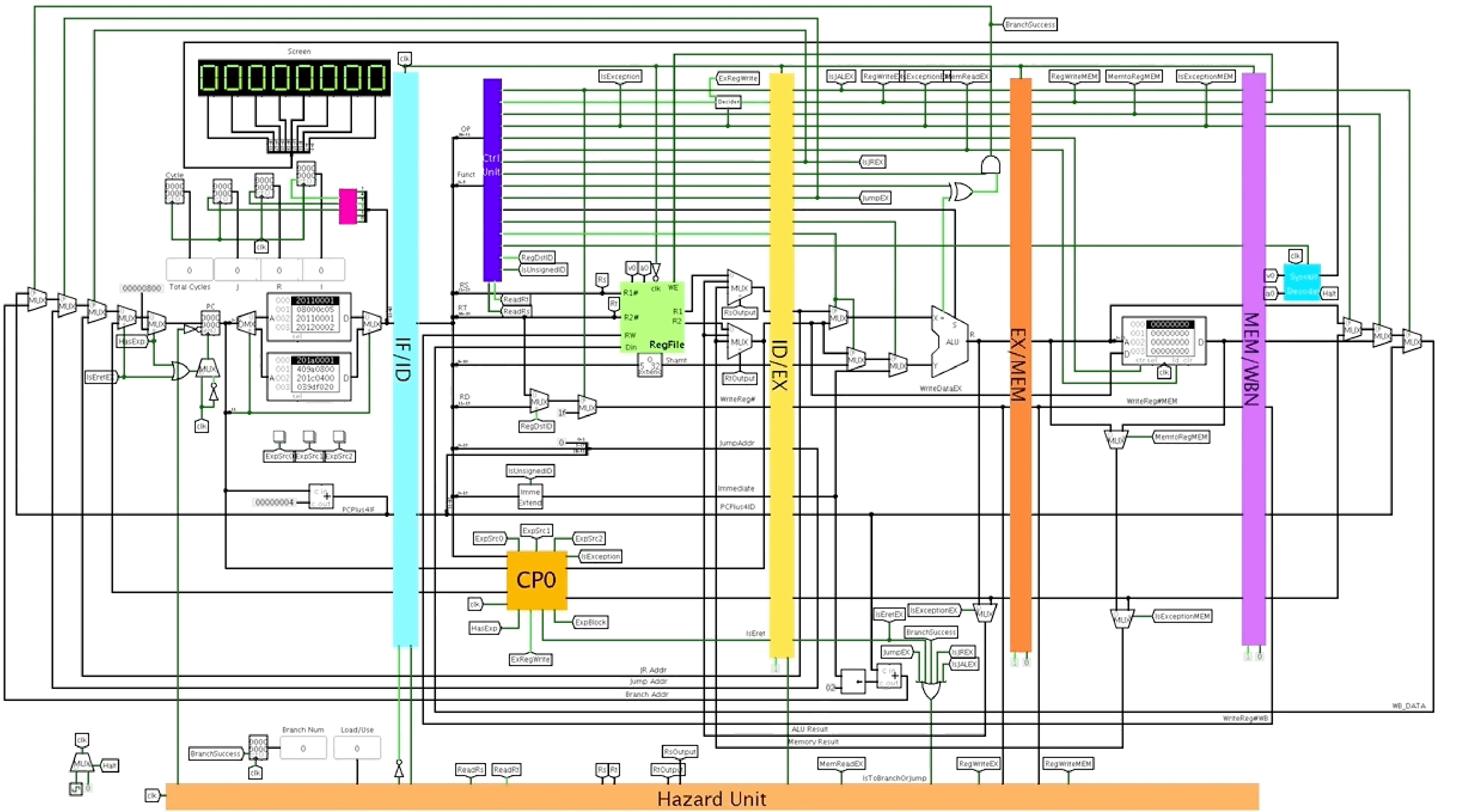
- 异常，中断
  - 顺序语义（进程序）：之前的指令完成，之后的指令未开始
    - ppl：在异常响应前，已经改变了系统的部分状态？
    - 面向ppl的ISA：“每条指令只写一个结果，且在WB段”，保证其他段不会提前改变系统“状态”！
  - 同步，异步
  - 断点与返回点：EPC
- 响应过程：软硬协同
  - 入口地址：非向量式（Cause），向量式
  - 现场保存与恢复
- 硬件控制最小化：EPC，Cause，flush/nPC
  - *单周期*
  - *多周期：中断周期*
  - 按序流水线：两个cc（当前周期发现，下一周期分支）
    - 断点：精确，非精确
    - 响应顺序：异常按序（进程序），中断按优先级
    - 各段的异常在本段处理（分布式）？提交点（集中式）？
- 铁律：程序执行时间=指令数×CPI×周期长度



PH processor： Pont@Univ of Leicester



仿真器☺



Screen
Ctrl Unit
RegFile
ALU
CP0
IF/ID
ID/EX
EX/MEM
MEM/WBN
Hazard Unit
Total Cycles
Branch Num
Load/Use
BranchSuccess
IsException
ExRegWrite
JumpEX
IsJREX
PCPlus4IF
PCPlus4ID
Immediate
JumpAddr
WriteReg#
WriteDataEX
WriteReg#MEM
MemtoRegMEM
IsExceptionMEM
JR Addr
Jump Addr
Branch Addr
ALU Result
Memory Result
WB_DATA

思考，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？
- 异常可精确或非精确，中断非精确？
- 流水线是否存在“中断周期”？
- EPC和cause应该在哪个段？异常检测电路？
- 为何提交点是M段？
- RV异常返回指令eret如何实现？
- 异常与中断同时发生，优先级？
- **取指、ID(非法指令)或访存M段出现异常咋办？**
- 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，参数传递，控制转移？

- 作业：4.30