异常与中断

异常并不意外！——llxx☺

llxx@ustc.edu.cn

程序异常：“未处理异常”

```
/*---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);
}
```

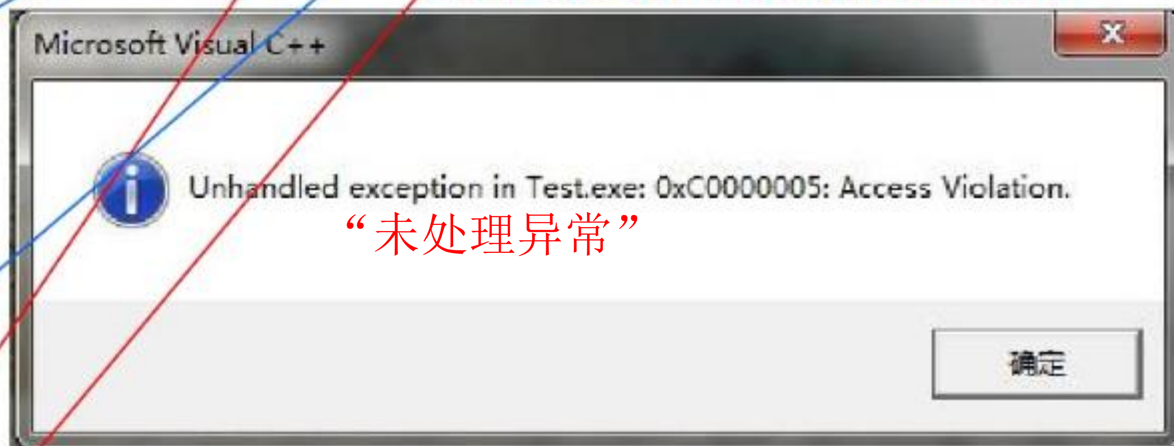
数据的表示

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）

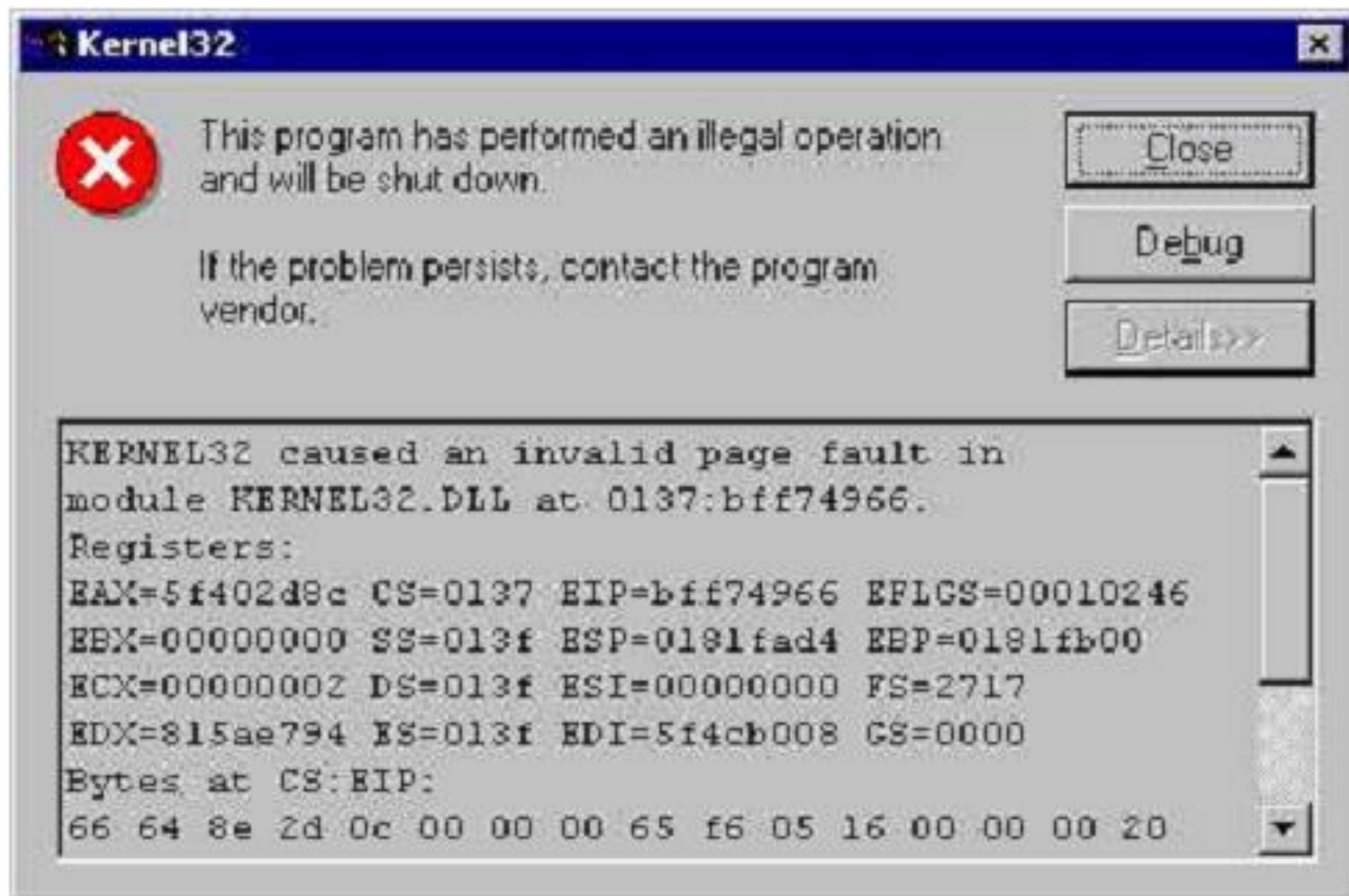
异常和中断处理

输入输出(I/O)

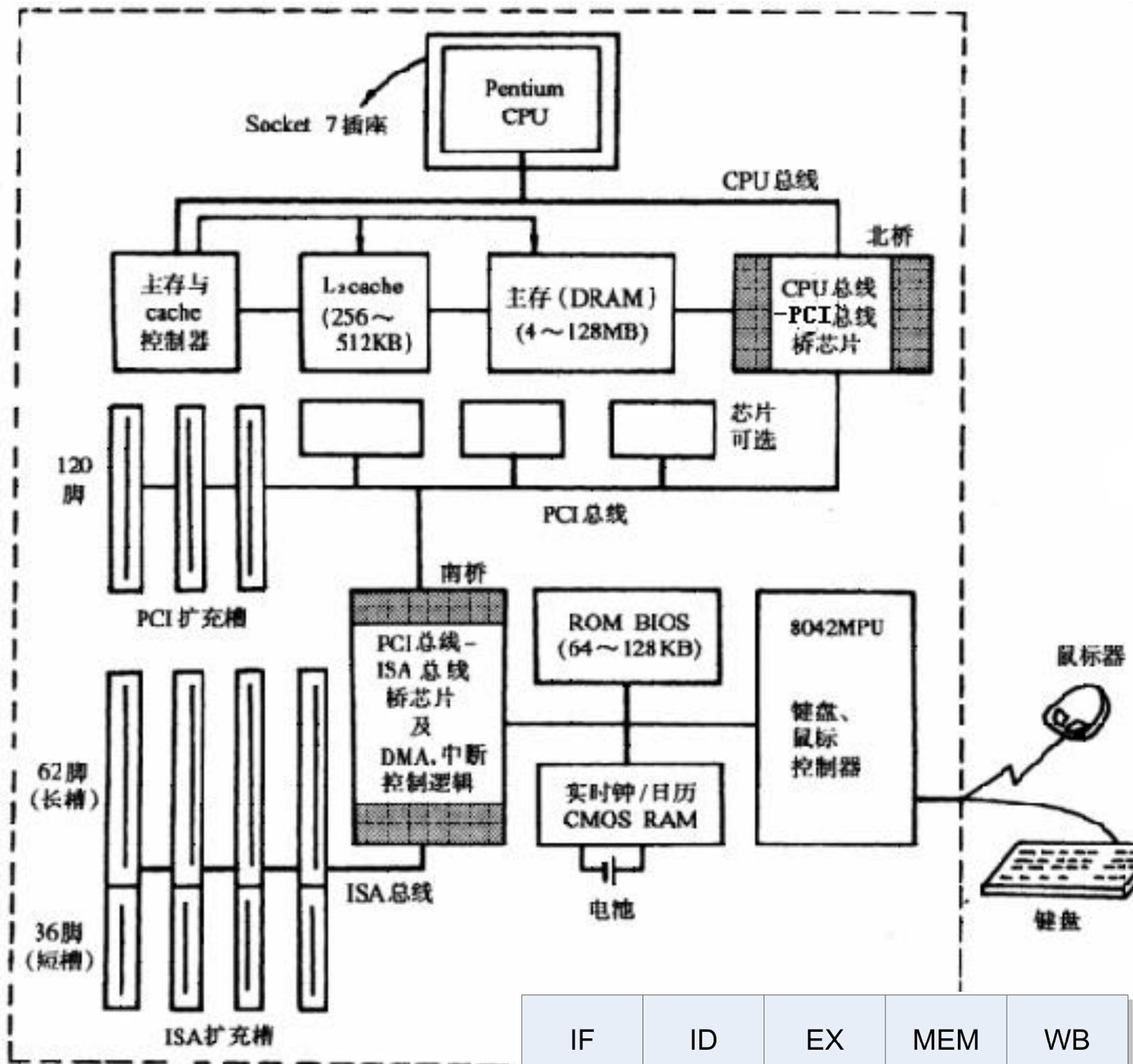
- 越界？异常点？

IF	ID	EX	MEM	WB

不可恢复异常：异常终止



输入输出

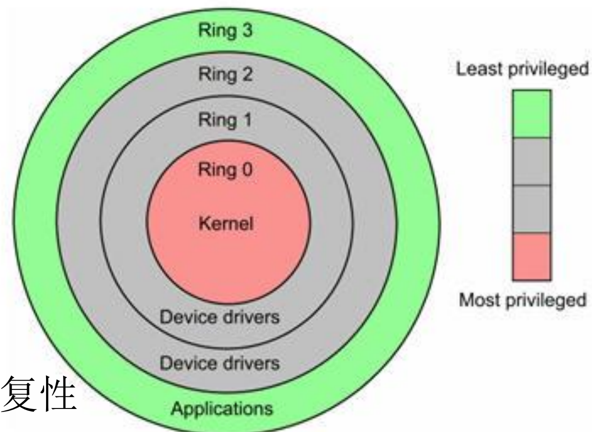


唐图10.19

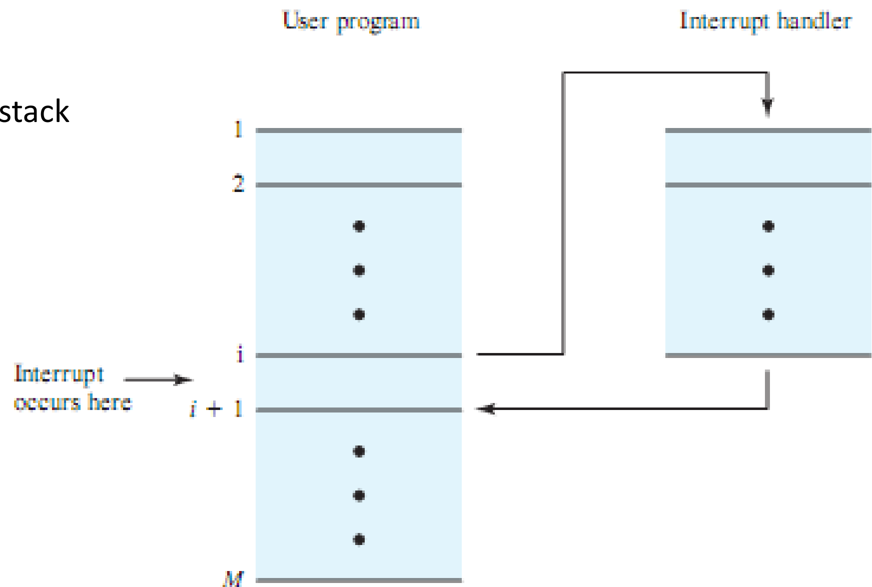
IF	ID	EX	MEM	WB

内容提要

- 异常与中断的基本概念，RV$4.10，唐5.5、8.4
  - 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（线程序）语义，发生时刻，响应时刻
    - 指令周期：同步，异步（中断周期）
  - 5种属性：同步性，主被动性，可屏蔽性，指令内外，可恢复性

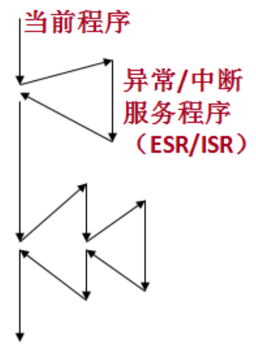


- 异常与中断响应的基本过程
  - 机构
    - ESR/ISR入口（向量式/非向量式），控制栈stack
    - 异常：EPC，Cause，Vector
    - 中断：EPC，Cause，Vector，Enable，Mask
  - OS：ESR/ISR
  - 约定：软硬件接口
- 异常与中断响应过程控制
  - 单周期：并发？
  - 多周期：异常，中断周期
  - 流水线：精确，非精确



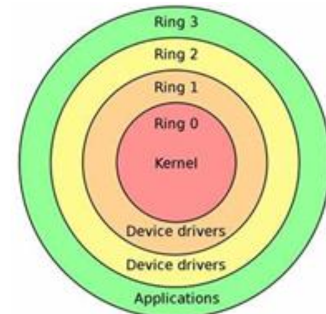
Terms：exceptions、interrupts、traps

- 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例程routine；
  - ESR/ISR执行完成后，被中断的程序可能恢复执行。
  - 与过程调用的区别：发生场景（程序内/外），系统状态切换
- 触发事件：RV/MIPS/ARM：异常/中断，x86：软中断/硬中断/异常
  - 内部事件：异常（例外），同步
    - 指令异常：非法指令、算术溢出/除零、访存缺页
    - 系统调用：服务（syscall，软中断/陷阱）、断点break
  - 外部事件：中断Interrupts，硬中断，异步
    - I/O（键盘，可屏蔽），硬件故障（存储器，不可屏蔽）



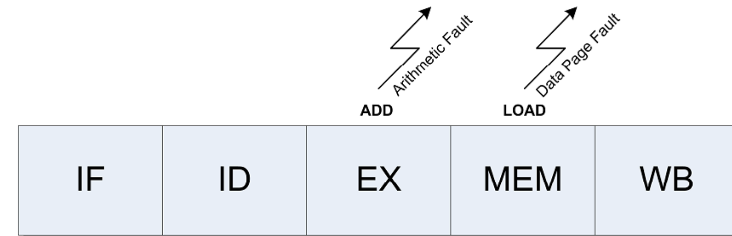
p235

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



指令异常与外部中断的发生时刻：同步性

- RV$6.10【电子版】："An I/O interrupt is **asynchronous** with respect to the instruction execution."
  - That is, the interrupt is not associated with any instruction and does not prevent the instruction completion.
  - The control unit needs only to check for a pending I/O interrupt at the time it starts a new instruction.

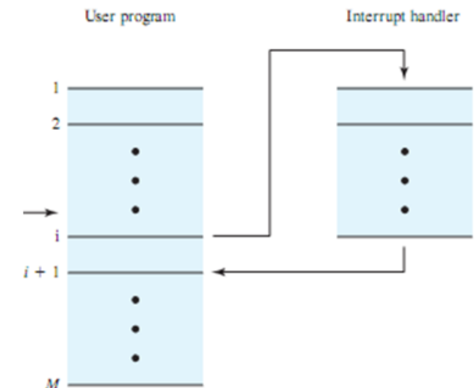


- f = (g + h) - (i + j); EX段ALU， sync

```
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)
```

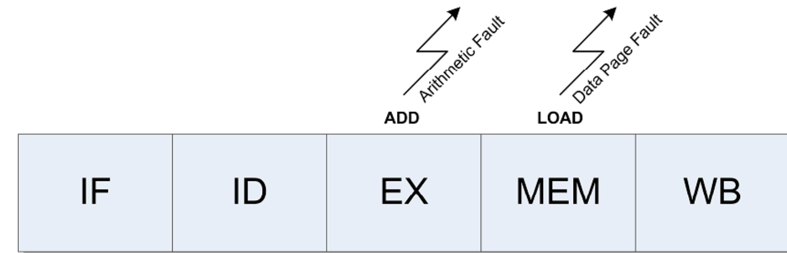
- A[12] = h + A[8]; MEM段MM， sync

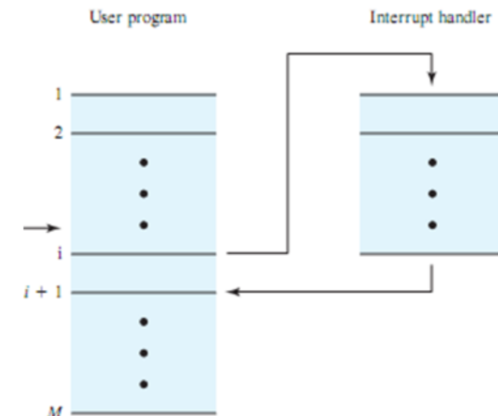
  ld x9, 64(x22) // Temporary reg x9 gets A[8]
  add x9, x21, x9 // Temporary reg x9 gets h + A[8]
  sd x9, 96(x22) // Stores h + A[8] back into A[12]



异常/中断的响应时刻：精确性

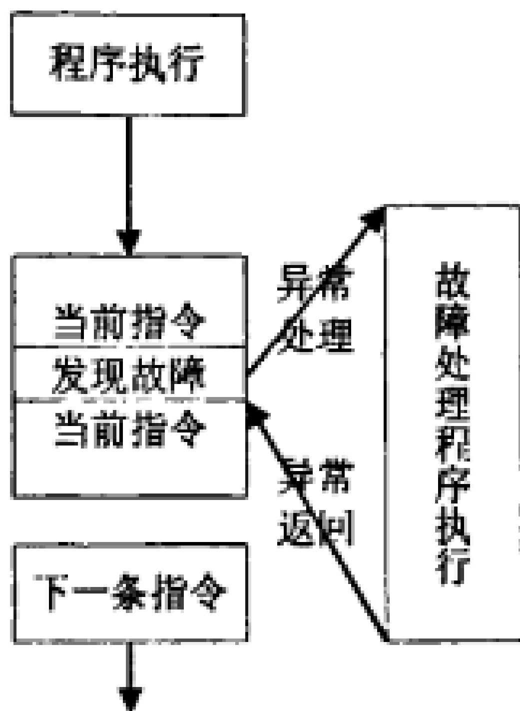
- 精确断点：按程序序（线程序）顺序执行，p239
  - 之前的指令已执行完成
    - 已经提交其体系结构可见状态
  - 之后的指令还没有发射
    - 没有改变任何机器状态
  - 精确性：当前指令，下一条指令
- 异常：立即响应，同步，精确
  - 指令异常：重启当前指令，或终止执行
  - 系统调用：返回下一条指令
- 中断：指令周期结束响应，异步，非精确
  - 非精确响应：哪一条指令？
    - 返回下一条指令，或“终止执行”



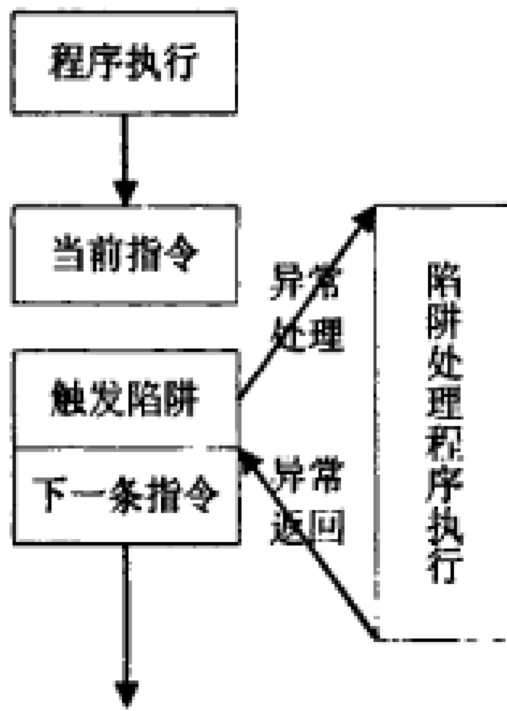


指令异常（故障）响应模式

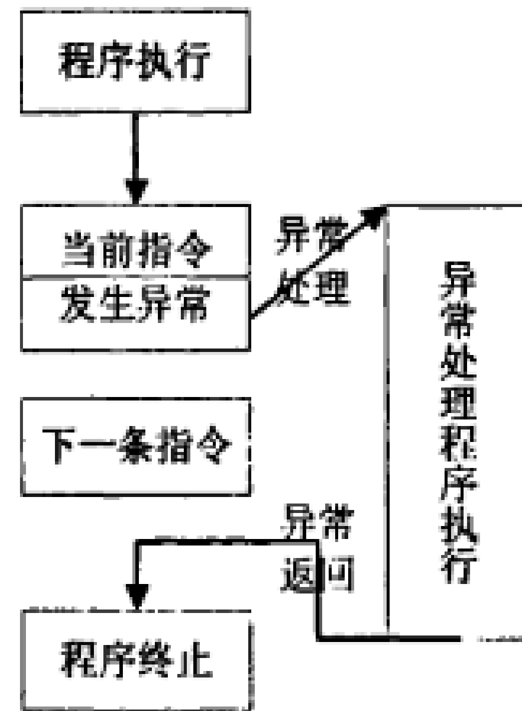
- 可恢复异常：访存缺页，OS处理后重新执行当前指令，如图“故障处理”
- 不可恢复异常：非法指令，交用户处理，如图“异常中止”
- 陷阱/软中断/系统调用sysCall：返回下一条指令



故障处理



陷阱处理



异常中止

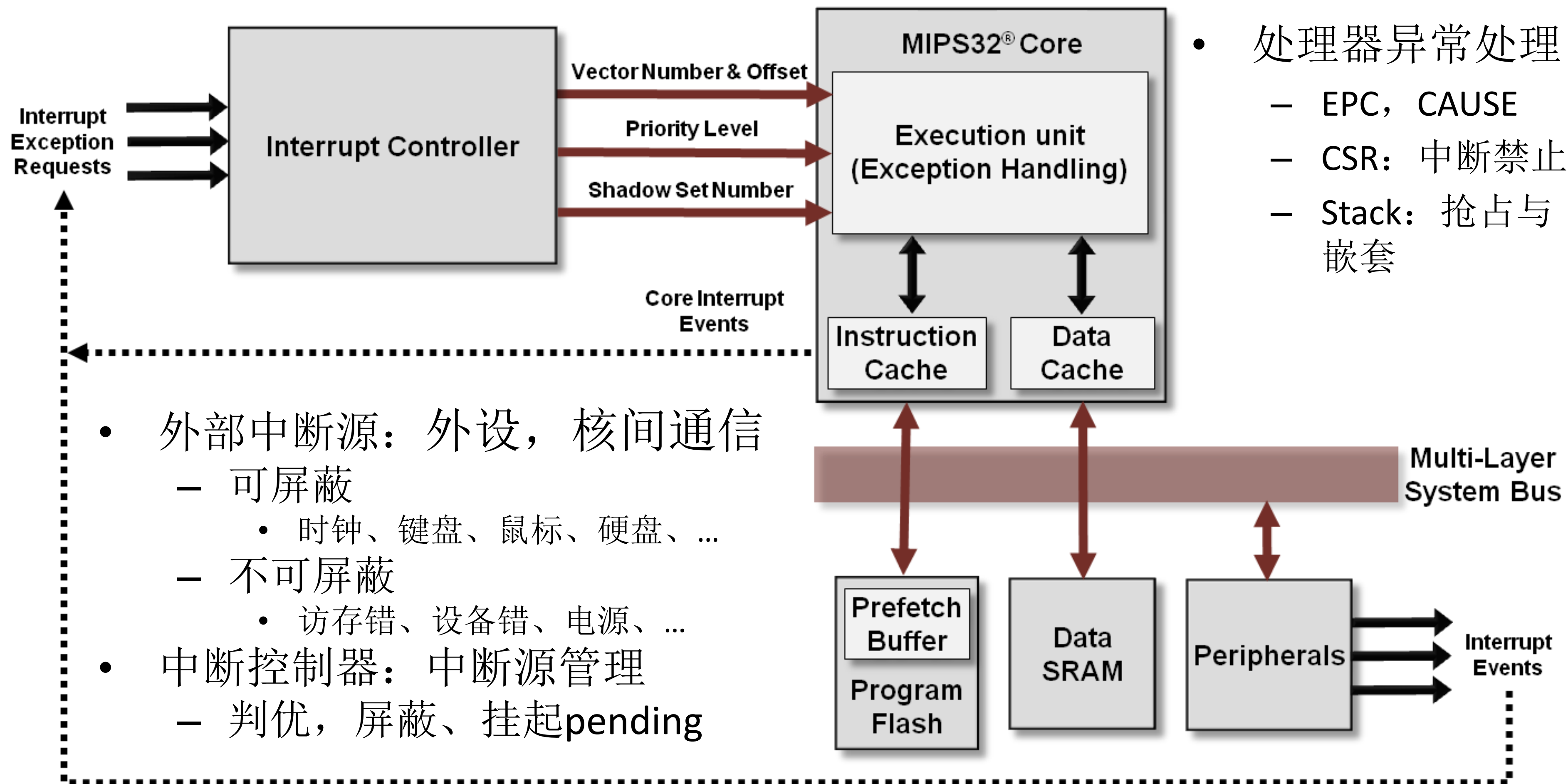
RISC异常/中断的属性：5种

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

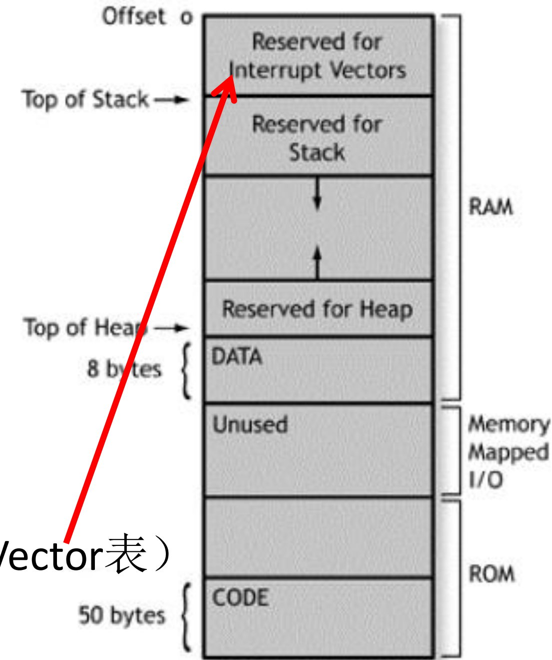
IF	ID	EX	MEM	WB

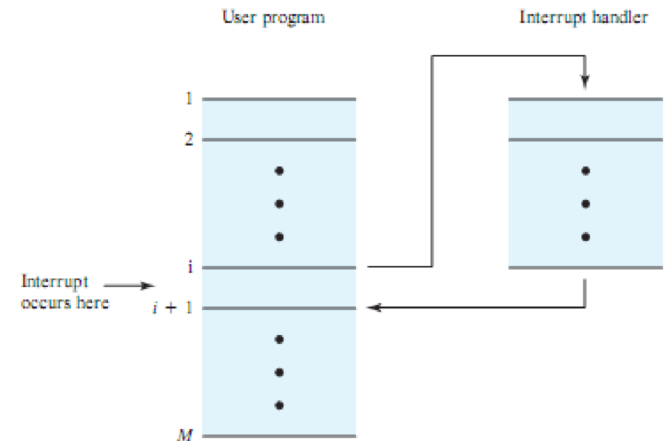
外部中断：中断源，类型，处理机构

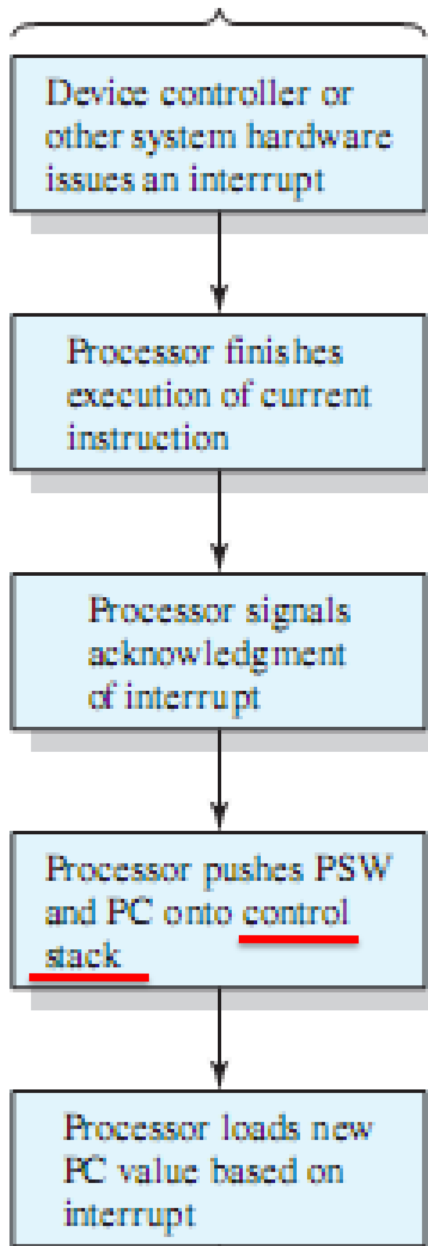


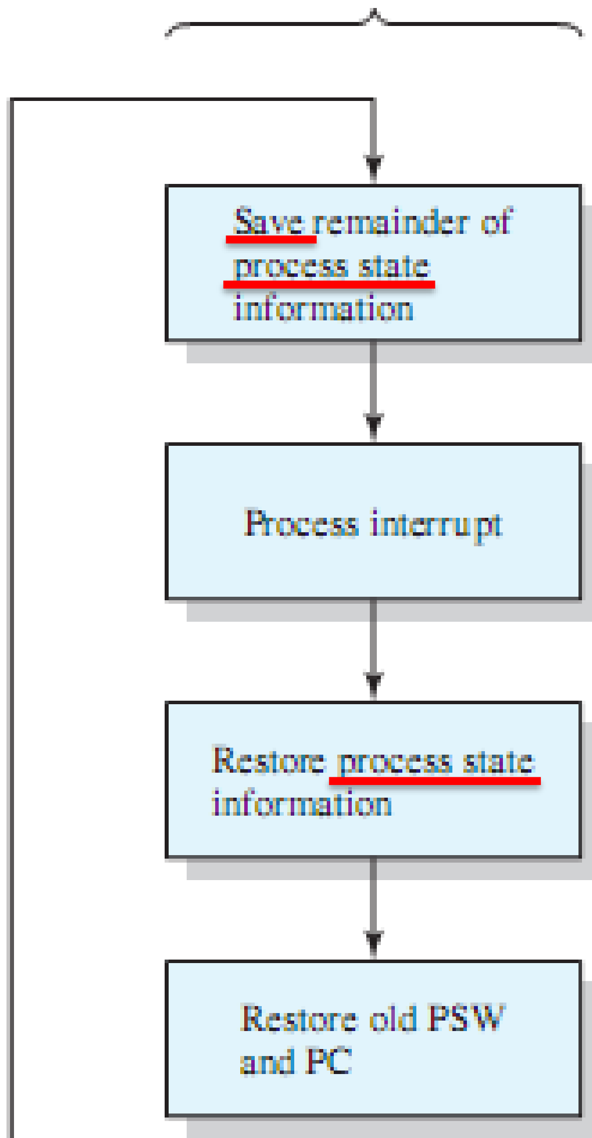
The Basics of Exception Handling

- 软硬件接口约定（$4.10.1）
  - 硬件：检查，记录，转ESR/ISR
    - 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：stack
    - *建立处理环境*：***user mode => kernel mode***
    - 转ESR/ISR入口：非向量式（统一入口）/向量式（Vector表）
  - 软件：服务
    1. 保存剩余现场（RF）：stack
    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
       - PC = 断点







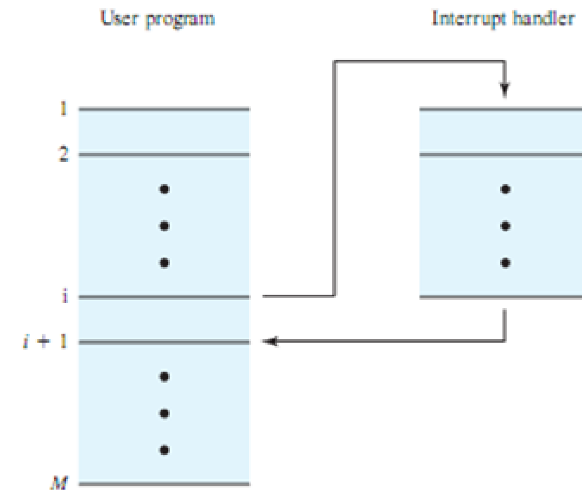


William Stalling《OS》

中断处理：

软硬件接口

约定



RV$4.10.1

ESR/ISR入口（非向量式）

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》

T − M
Control stack
T
Y
Start
Interrupt service routine
Y + L
Return
N
N + 1
User's program
Main memory
Y
N + 1
Program counter
General registers
T
Stack pointer
Processor
T − M

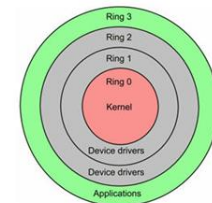
保存断点与现场

T − M
N + 1
Control stack
T
Y
Start
Interrupt service routine
Y + L
Return
N
N + 1
User's program
Y + L + 1
Program counter
General registers
T − M
Stack pointer
Processor
T

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（存储映射I/O）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

访问CSR指令，$5.14

- CSR swap instructions
  - 6条：I-type，12位=4K
  - *read-modify-write* cycle
    - 将CSR复制到通用Reg，用寄存器操作数(CSRRW)或立即数(CSRRWI)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

ESR/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：——可维护性，减少代码量

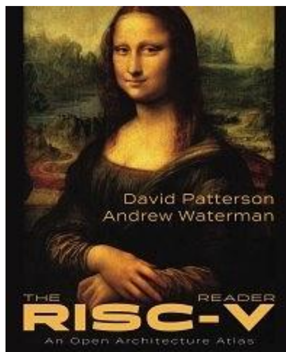
- **a/t**-registers are caller-saved
- **s**-regs are callee-saved and **preserve** their contents across function calls

TimerISR

非向量式中断

图10.6

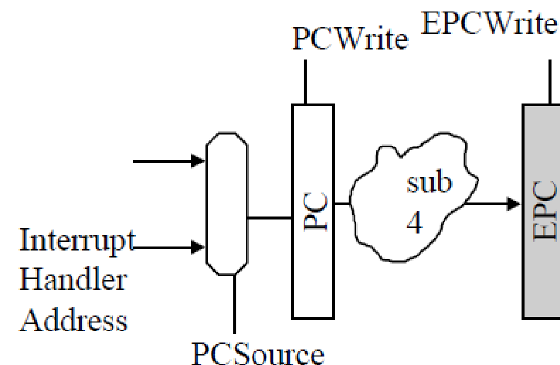
```
# save registers
csrrw a0, mscratch, a0   # save a0; set a0 = &temp storage
sw a1, 0(a0)             # save a1
sw a2, 4(a0)             # save a2
sw a3, 8(a0)             # save a3
sw a4, 12(a0)            # save a4

# 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
```

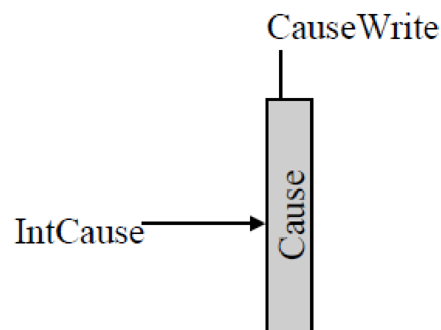


David Patterson，Andrew Waterman，The RISC-V Reader: An Open Architecture Atlas，2017

CPU指令异常处理过程：IIxx四步法

- 过程分解：异常响应的4个硬件微操作（EPC、Cause、转Handler、ERET）
  - *异常触发：与当前指令同步。同一周期多个异常？*
  - 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）： IntCause = 0
    - 算术溢出（在EXE周期）：IntCause = 1
    - *缺页*
  - *保存控制状态字寄存器CSR*
  - 转ESR： 入口00 0010 0000（RV$4.10.1），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、控制器、综合：单周期、多周期、PPL
  - 部件：EPC，Cause，入口地址/Vector表地址
  - 控制信号：EPCWrite，CauseWrite，PCSource/PCWrite，Flush，时序





单周期异常：数据通路、控制、时序
发生多个异常？中断？

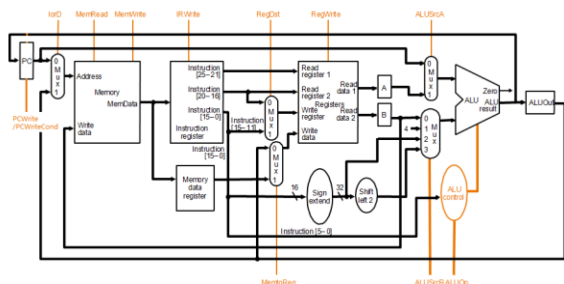
Control Unit
MemtoReg
MemWrite
Branch
ALUControl$_{2:0}$
ALUSrc
RegDst
RegWrite
Op
Funct
PCSrc
PCWrite
EPCWrite
Interrupt Handler Address
PCSource
PC
sub 4
EPC
31:26
5:0
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
Zero
ALU
ALUResult
WriteData
WE
A
RD
Data Memory
WD
ReadData
20:16
15:11
WriteReg$_{4:0}$
PCPlus4
4
15:0
Sign Extend
SignImm
<<2
PCBranch
CauseWrite
IntCause
Cause
Result

多周期异常

PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op [5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
npc
Instruction [25–0]
26
Shift left 2
28
Jump address [31-0]
PC [31-28]
PC
Mux
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
Zero
ALU
ALU result
ALUOut
beq
Memory data register
16
Sign extend
32
Shift left 2
ALU control
Instruction [5–0]

多周期控制器FSM

每个状态一个时钟周期。



Instruction fetch

0

MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00

Start

Instruction decode/
register fetch

1

ALUSrcA = 0
ALUSrcB = 10
ALUOp = 00

(Op = 'LW') or (Op = 'SW')

(Op = R-type)

(Op = 'BEQ')

Memory address
computation

2

ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00

Execution

6

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10

Branch
completion

8

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 1

(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

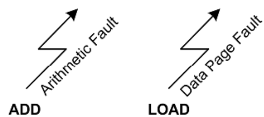
Write-back step

4

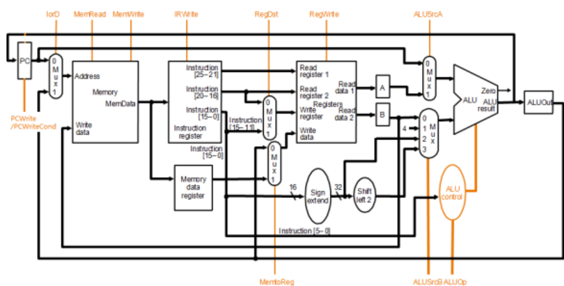
RegDst = 0
RegWrite
MemtoReg = 1

指令异常：

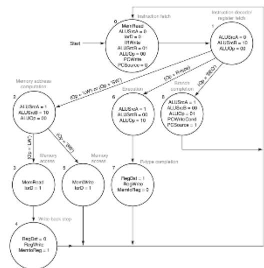
- 非法指令
- 算术溢出
- *缺页*



IF	ID	EX	MEM	WB



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

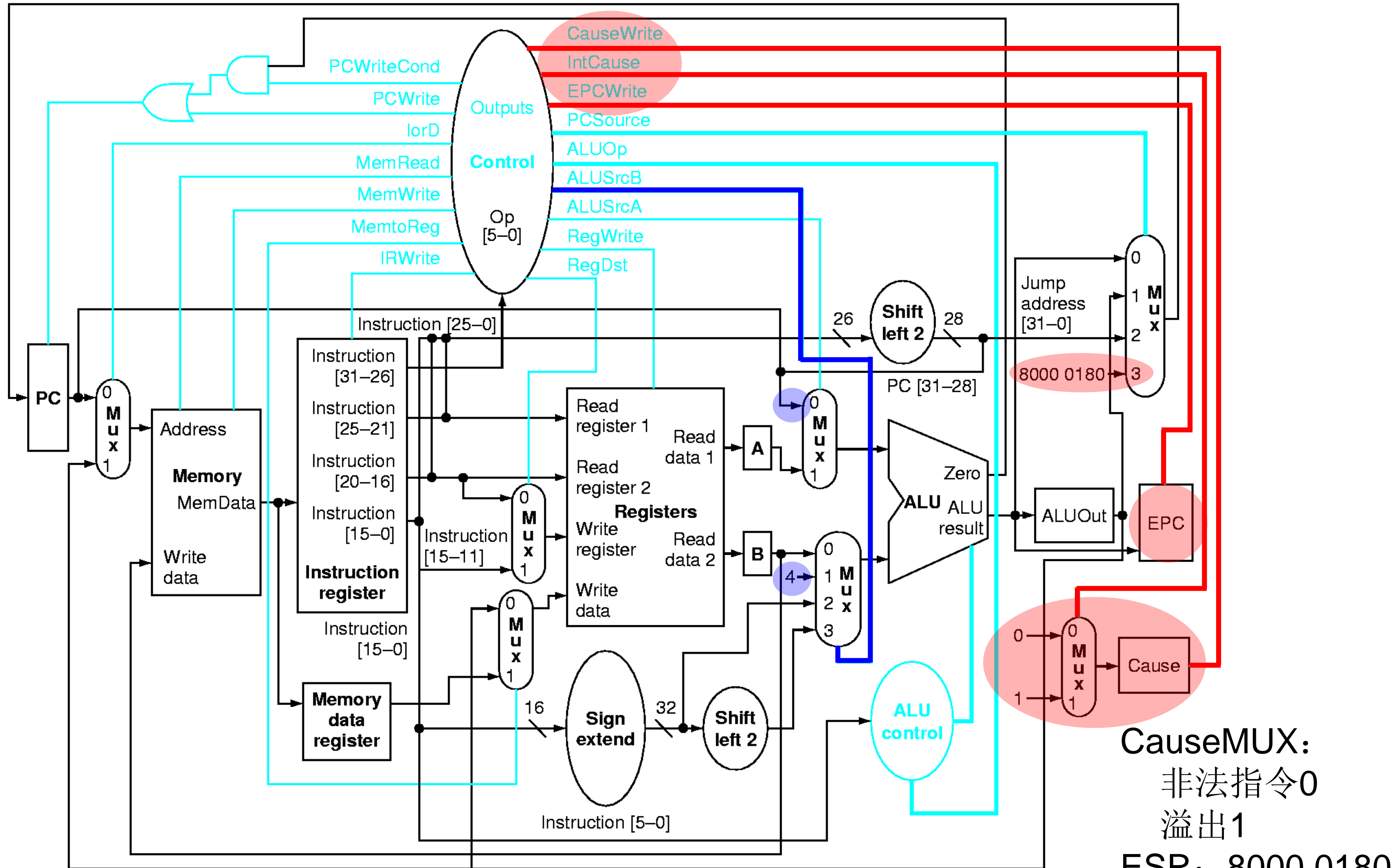
4
RegWrite
MemtoReg = 1
RegDst = 0

$\overline{\text{Overflow}}$

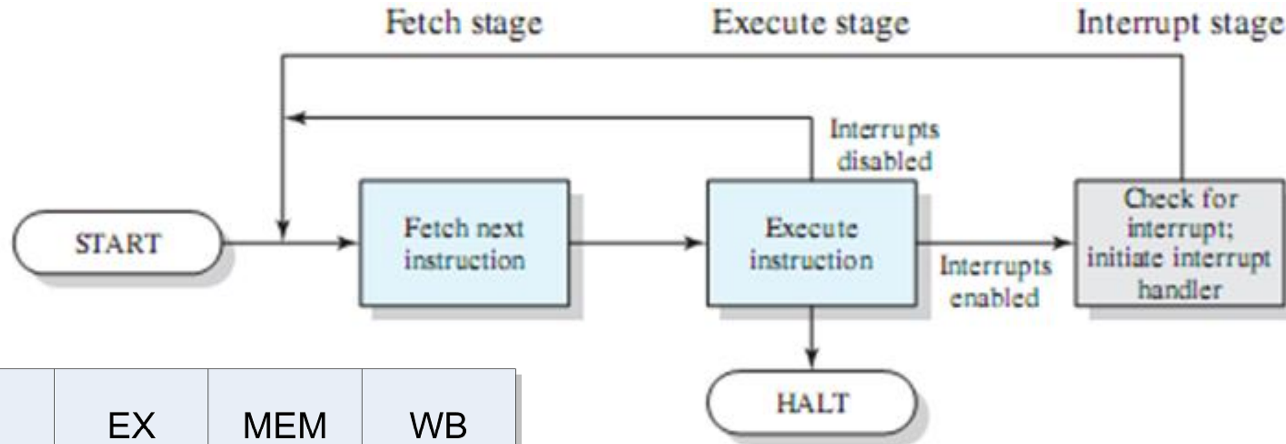
溢出时，先写回了RF！　　ALU：PC-4 => EPC

COD3 图5-40

Exceptions Handling in Multi-Cycle MIPS

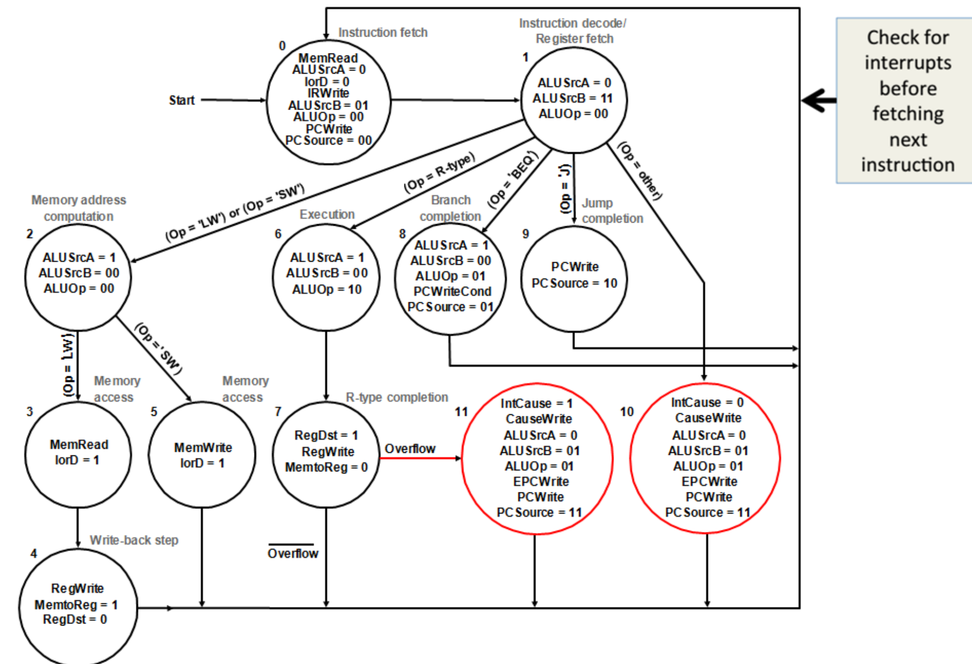


Multicycle Interrupts：中断周期

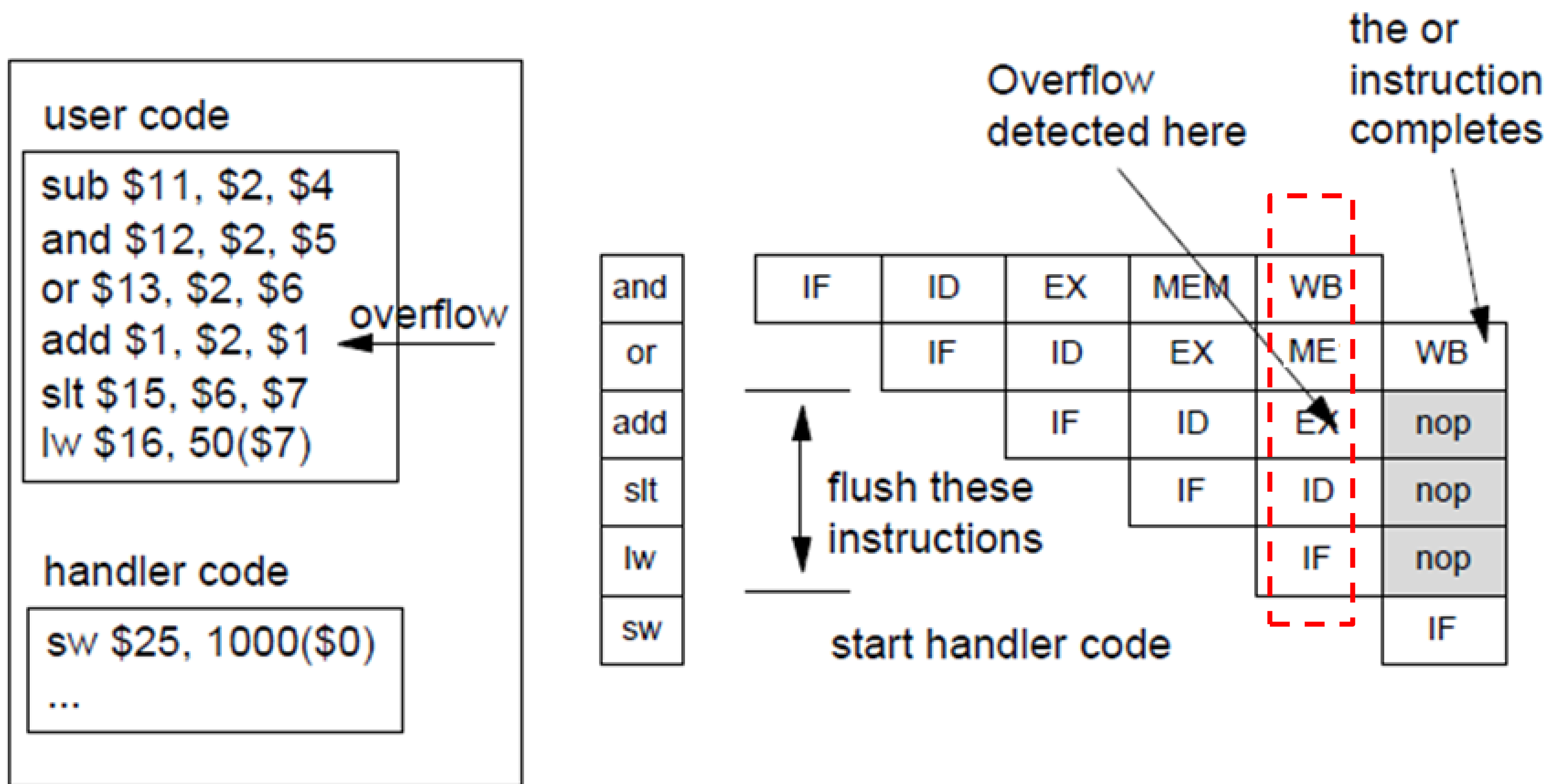


IF	ID	EX	MEM	WB

- 中断控制器IntrCtrler
  - Mask，判优
- CPU
  - 条件：中断Enable
  - 检查intReq
    - 保存断点：EPC，Cause
    - 转ISR
- 异常与中断同时发生？

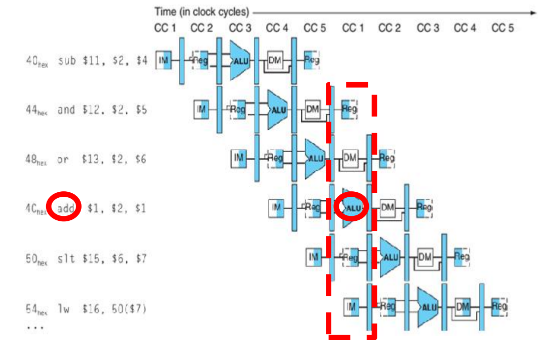


ppl异常处理：add溢出，$4.10.2



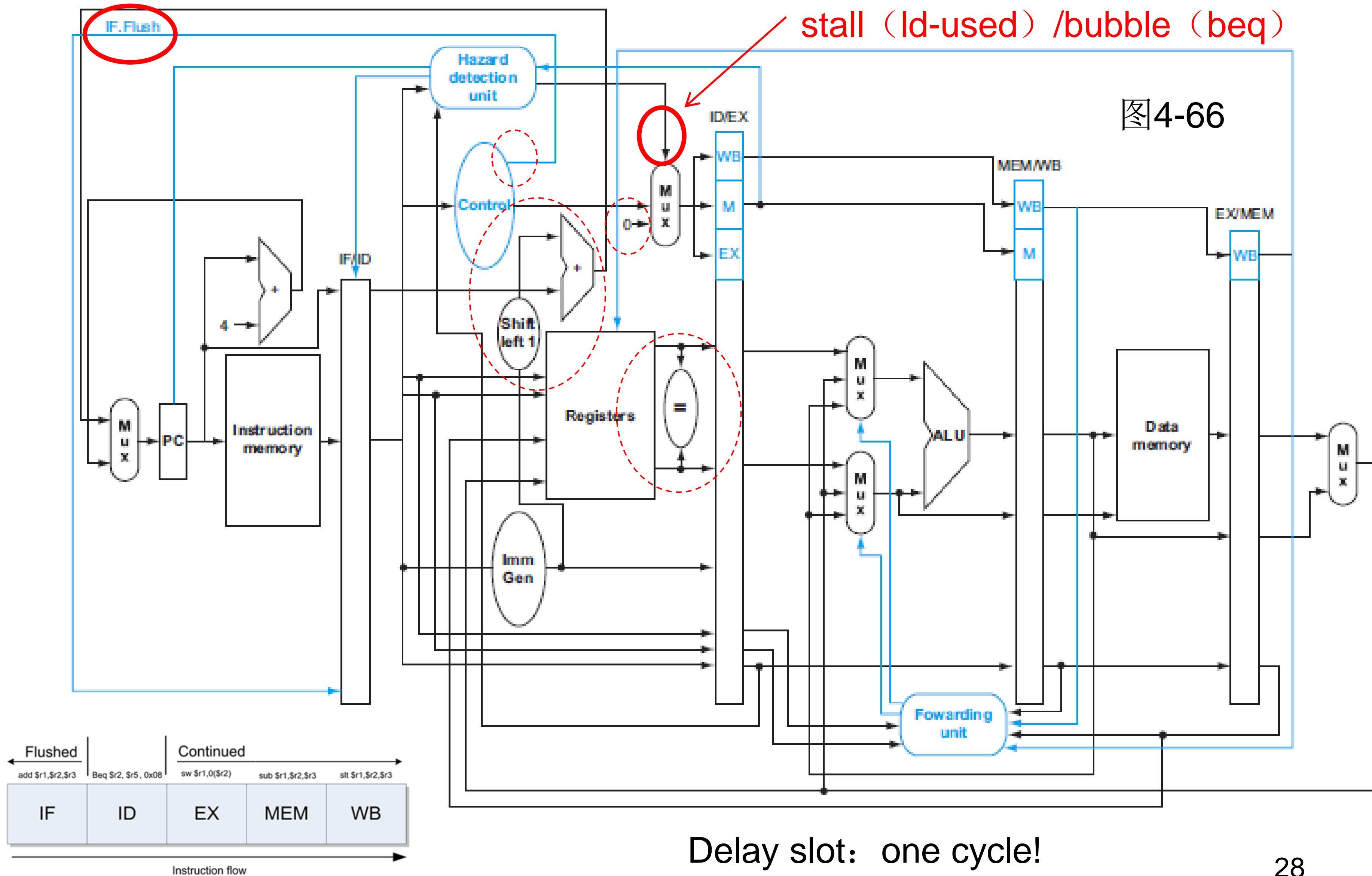
- 保证异常响应的顺序与指令执行的逻辑顺序相同
  - 顺序语义：之前的指令完成，之后的指令取消
  - 精确断点：EPC = add地址

RV ppl的add溢出异常处理$4.10.2



- 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【beq？】
    - add溢出：IF.Flush，ID.Flush，EX.Flush
    - "Flush signals to be set near the end of this clock cycle"
  - 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（终止），其他（缺页）可能继续（恢复）
- EPC/Cause放在哪个流水段？
- "简约原则"：硬件提供最少支持（EPC/Cause，分支）
  - 其他OS负责（分析Cause，ESR编程）

数据通路模版：单周期beq实现，IF.Flush，bubble

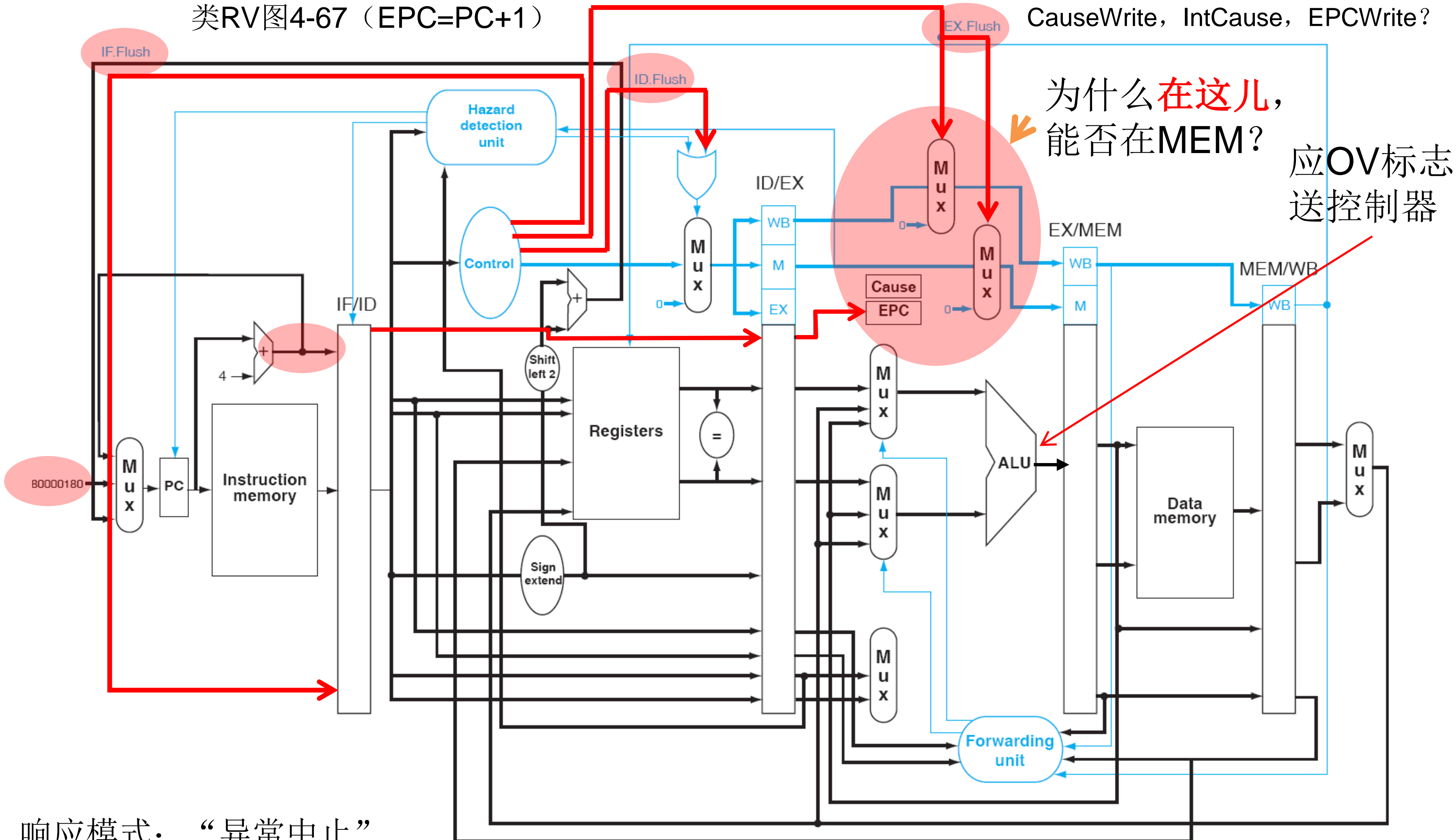


Delay slot：one cycle!

add指令overflow：flush IF/ID/EX

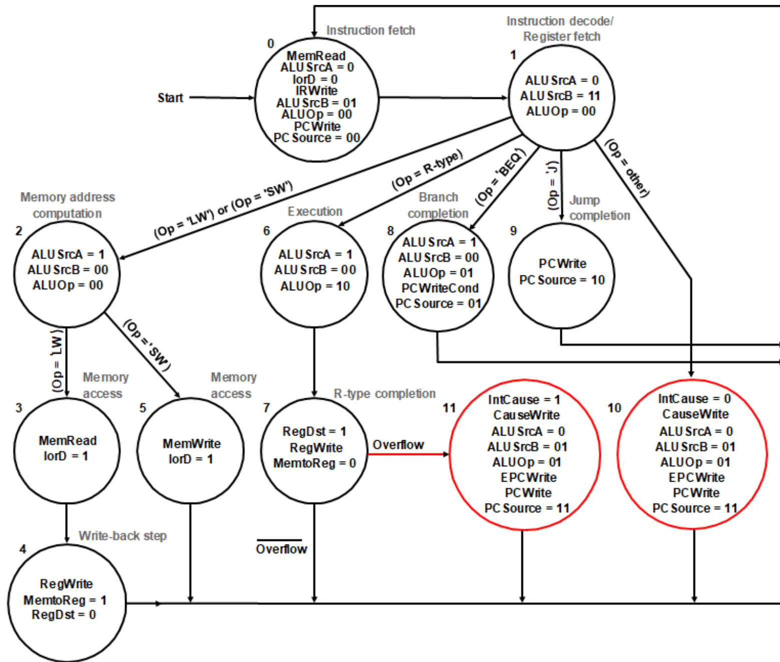
COD4图4-66
类RV图4-67（EPC=PC+1）

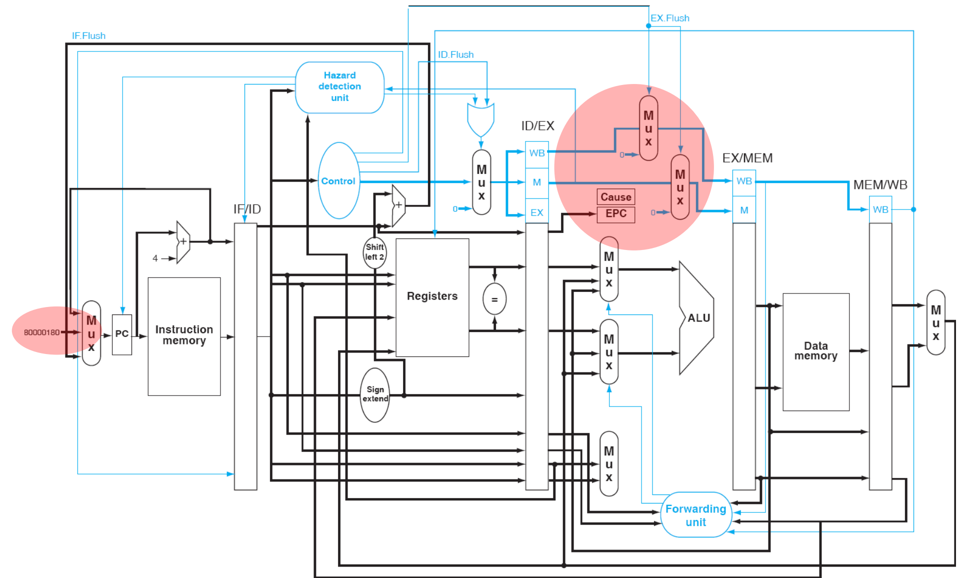
控制信号（多周期图）：
CauseWrite，IntCause，EPCWrite？



响应模式：“异常中止”
当前周期：检测，记录（EPC、CAUSE），排空IF/ID/EX，ESR入口写nPC
下一个周期：取ESR的第一条指令8000180

OV异常处理为什么在EX，能否在MEM？

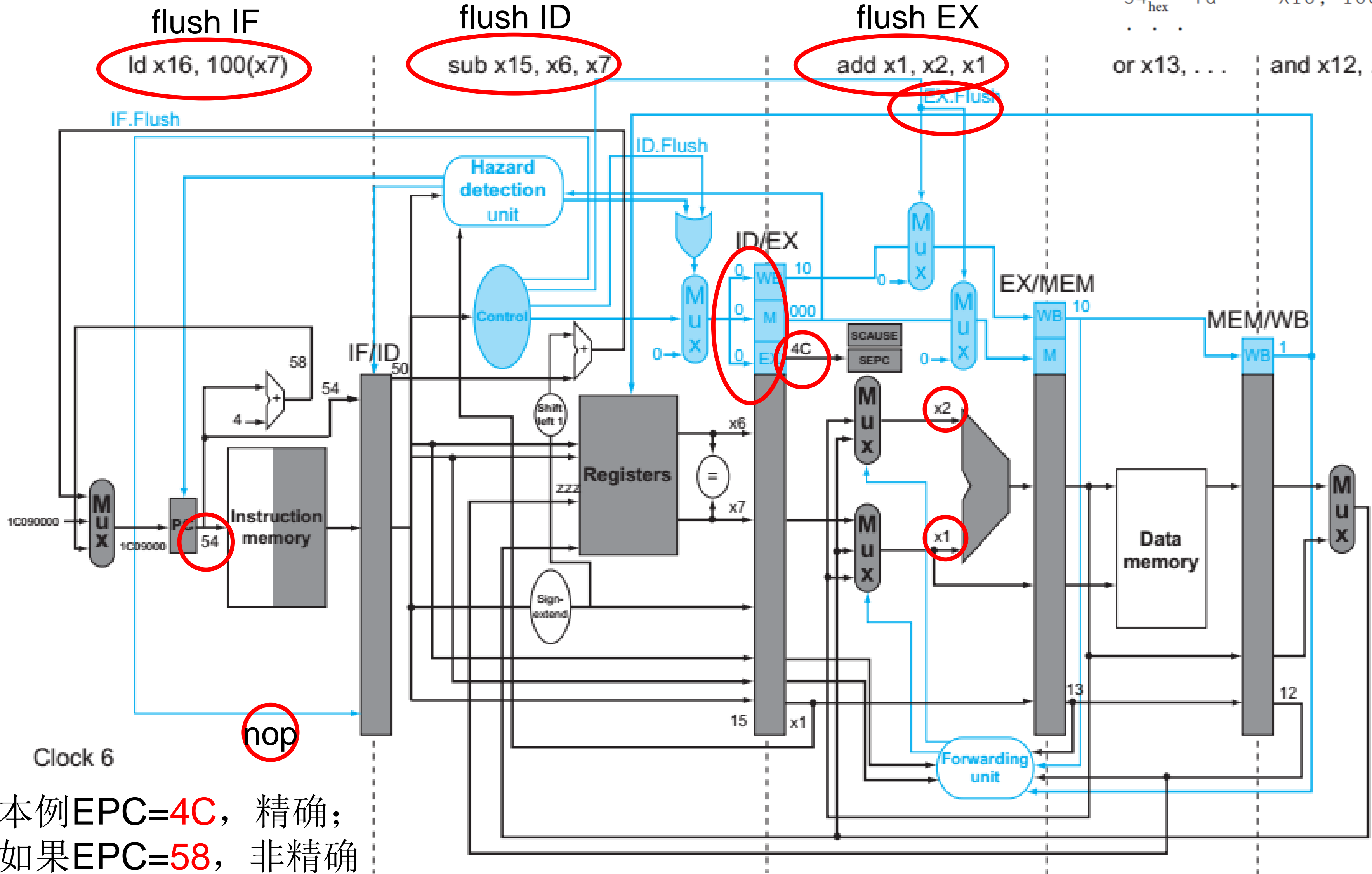




- ppl异常处理机构：EPC，Cause，Flush IF/ID/EX
- 提交点？

RV图4-68， 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)
. . .
```



发现，记录（EPC、CAUSE），排空IF/ID/EX，ESR入口写nPC

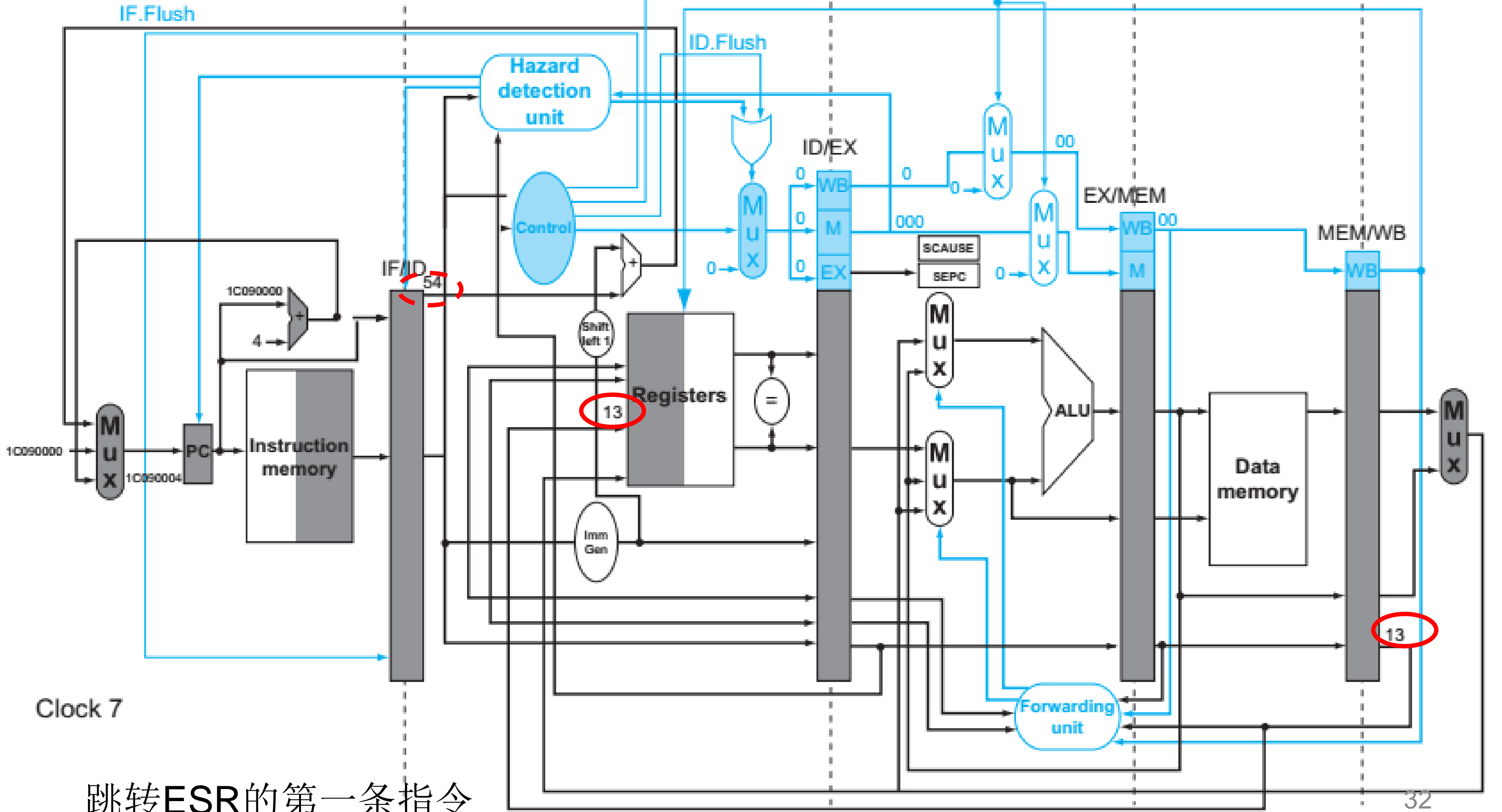
RV图4-68，CC7

```
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)
...
```

```
1C090000hex  sw  x26, 1000(x10)
1C090004hex  sw  x27, 1008(x10)
...
```

first instruction of exception routine

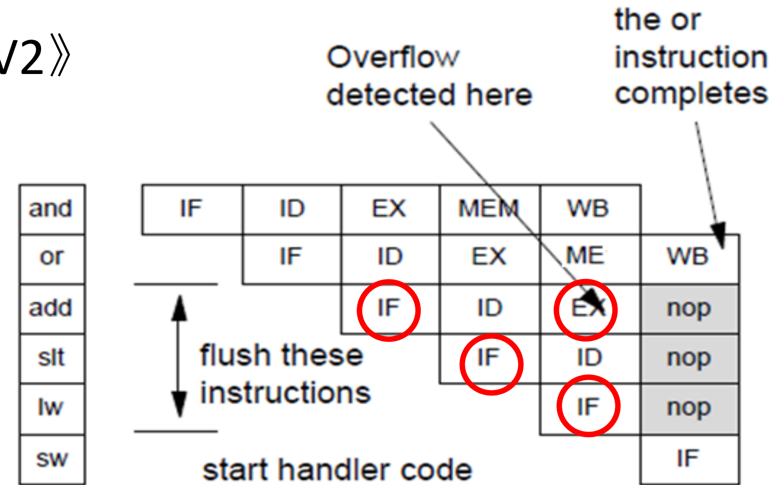
sd x26, 1000(x0) | bubble (nop) | bubble | bubble | or x13, . . .



跳转ESR的第一条指令

多指令多异常：了解，llxx

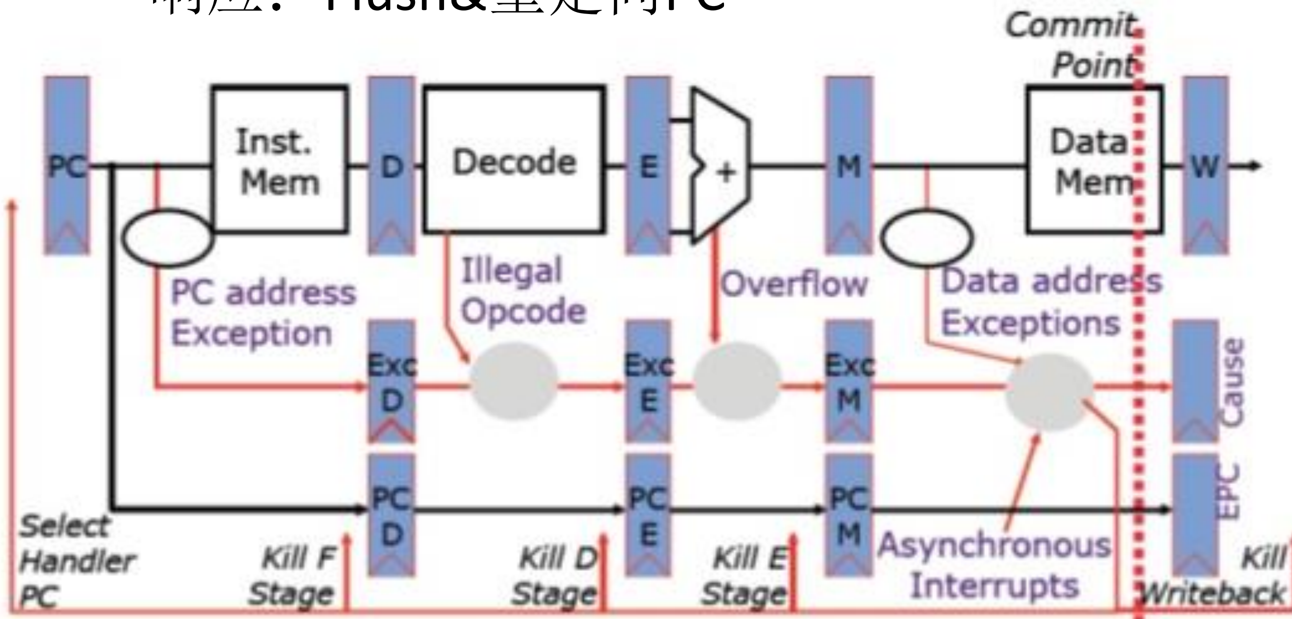
- 单指令单异常： add溢出（EX段），《RV2》
- 单指令多异常（不同段）： FCFS
  - IF段：缺页
  - ID段：非法指令
  - EX段：溢出/除零
  - MEM段：缺页
  - WB段：无（写寄存器错？）

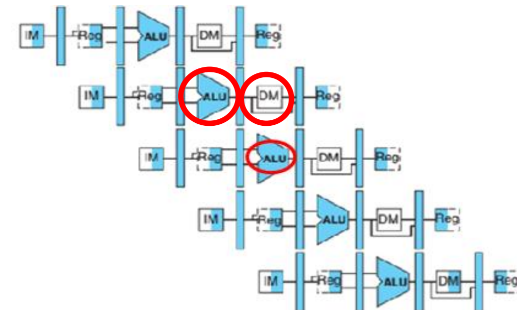


- 多指令不同周期异常：[slt@CC4，add@CC5]，FCFS
- 多指令同时异常（同一周期）：FCFS，优先级
  - 精确（RV）：流水线中的异常与引发异常指令精确关联
    - EPC=异常指令，如add
  - 非精确（DLX）：异常与执行不精确关联，如EPC=IF段指令
    - 硬件简单，但软件复杂：需要OS依据流水线深度确定EPC
- 异常的响应时刻：在本段立即处理？
  - 分布式：多套EPC/Cause，多段Flush，复杂☹
  - 集中式：提交点技术

多异常处理：集中式，提交点技术，精确

- 异常提交点：哪一段？
  - 出现异常并不立即响应，仅存EPC（背包）并kill（气泡），直至提交点
    - MIPS：提交点在MM段，EPC/Cause放在MM/WB段
    - Dubois书：提交点在WB段
  - 多指令多异常：响应流水线中最早的异常，kill新的异常
- 响应：Flush&重定向PC

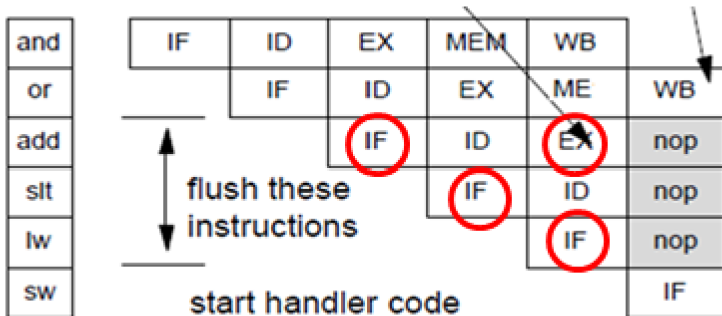


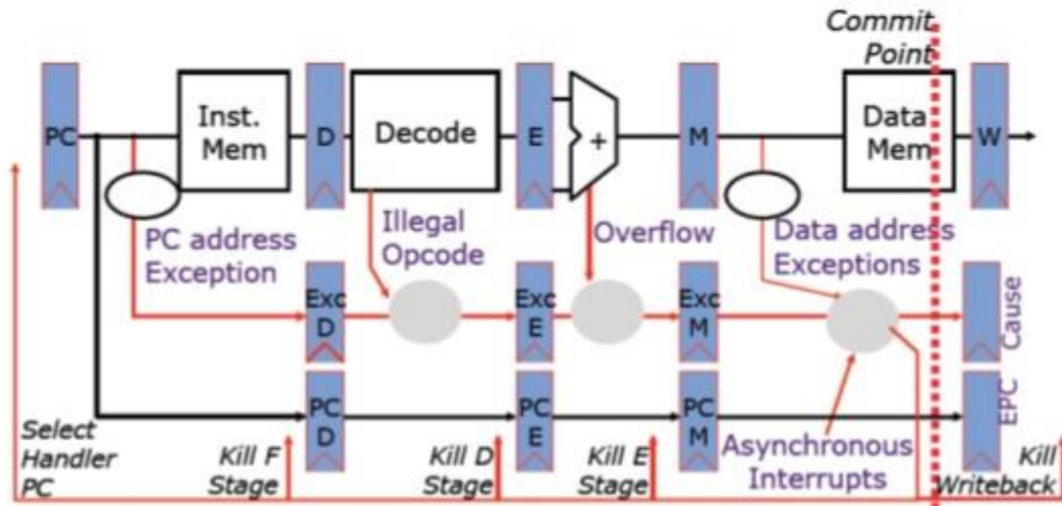


《Computer Architecture: A Constructive Approach》Fig11.3 $3.3.1

流水线中的中断响应

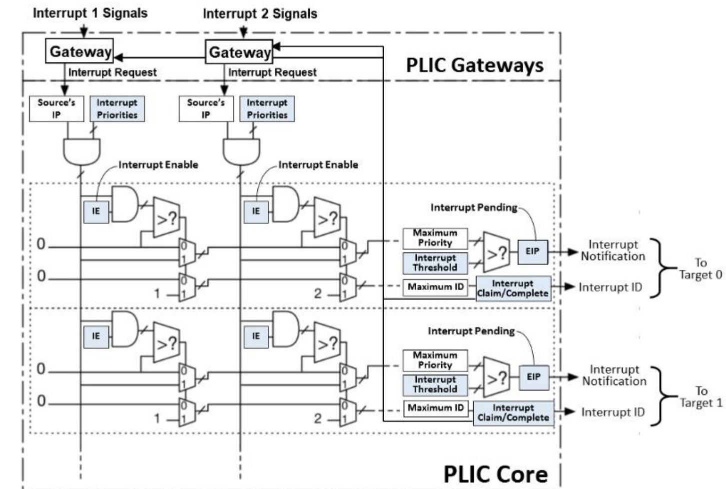
- 中断的非精确性：允许不精确！！！
  - $4.10.2：“I/O设备请求和硬件故障与特定指令无关，因此何时中断流水线具有较大的灵活性”【中文书的翻译不准确】
  - 中断响应：停止IF取指（EPC），完成已进入流水线的指令，转ISR
    - 出现异常？
- 提交点：外部中断注入点，按优先级
  - 中断：EPC = 注入点IF段的当前指令
  - 多中断：优先级，嵌套？
  - 异常+中断：优先级？

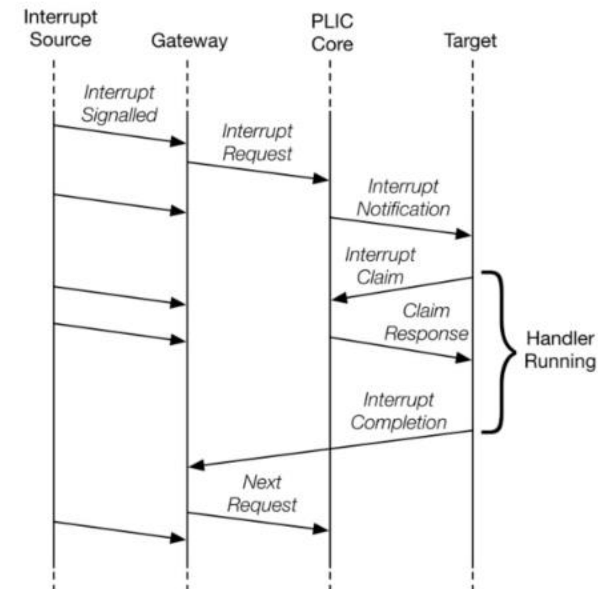




RV的中断机制（蜂鸟书第13章）

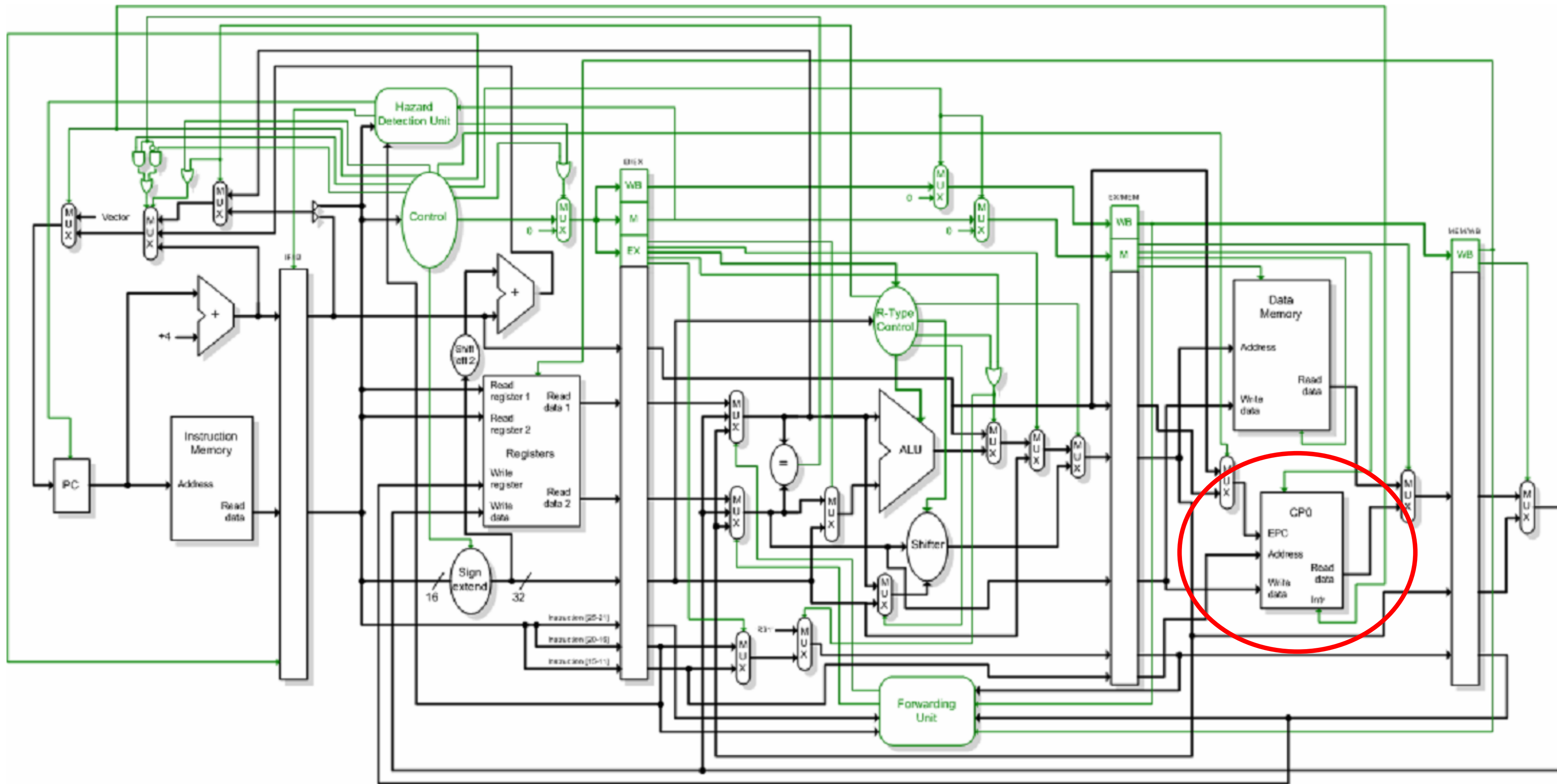
- 异常类型
  - 同步异常
  - 精确异步异常（外部中断）
  - 非精确异步异常：已经无法与特定指令关联
    - 如：长指令写回时的异常，Cache写回时访存错
- PLIC Gateways
- PLIC core：
  - 支持多中断目标（Hart）
    - Target按其所定义的优先级阈值响应中断
  - 进行外部中断源标识和仲裁
    - 每个中断源标识包括：Gateway（触发类型）、IP（Pending）、优先级、ID（同优先级）、IE（Enable）
  - 生成一个硬中断信号，送给合适的core/Hart
  - PLIC与target握手
    - 通知（PLIC->target ）、确认（target->PLIC）、完成（target->PLIC ）
- CLINT：core local interrupts controller
  - 蜂鸟：一个软中断信号，一个计时中断，送给CPU



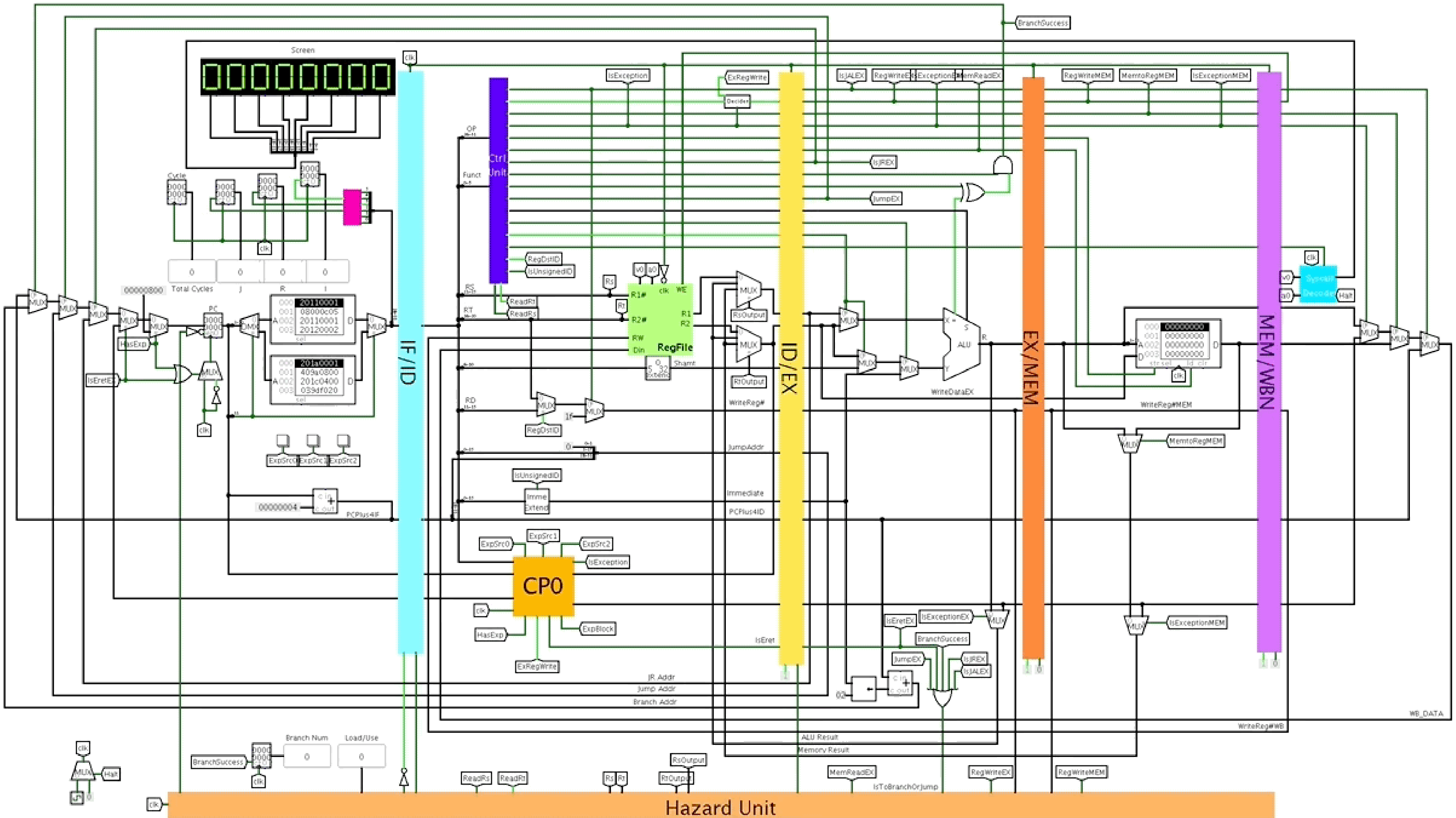


The RISC-V Instruction Set Manual Volume II: Privileged Architecture【2016，仅PLIC】

PH processor： Pont@Univ of Leicester

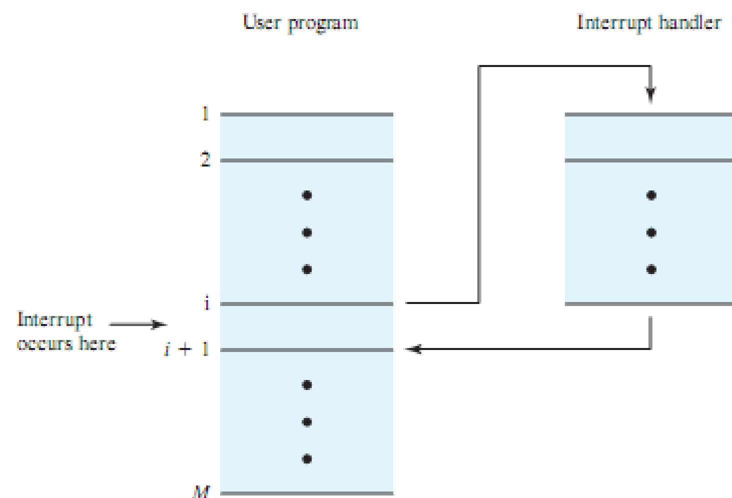


仿真器☺



小结

- 异常，中断：同步，异步
  - 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”！
  - 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



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

llxx@ustc.edu.cn