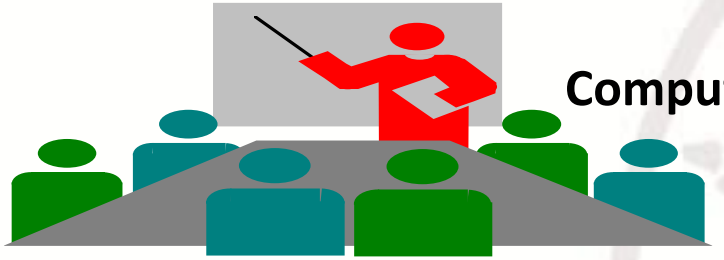


Computer Organization & Design
实验与课程设计

实验八

CPU设计-中断

施青松
Asso. Prof. Shi Qingsong
College of Computer Science and Technology, Zhejiang University
zjsqs@zju.edu.cn

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

实验目的

1. 深入理解CPU结构

3. 学习如何提高CPU使用效率

3. 学习CPU中断工作原理

4. 扩展设计简单中断

5. 设计中断测试程序

实验环境

实验设备

1. 计算机（Intel Core i5以上，4GB内存以上）系统

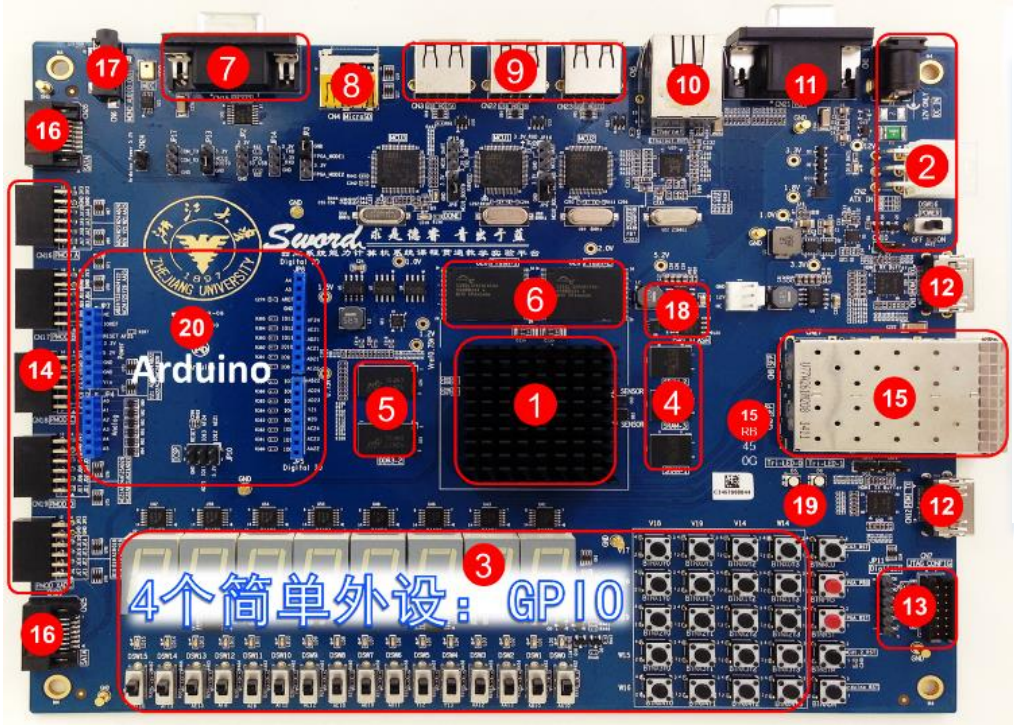
2. 计算机软硬件课程贯通教学实验系统(Sword)

3. Xilinx ISE14.4及以上开发工具

材料

无

计算机软硬件课程贯通教学实验系统



贯通教学实验平台主要参数

▼ **核心芯片**

Xilinx Kintex™-7系列的XC7K160/325资源：

162,240个，Slice：25350，片内存储：11.7Mb

▼ **存储体系** 支持32位存储层次体系结构

6MB SRAM静态存储器：支持32Data，16位TAG

512M BDDR3动态存储：支持32Data

32MB NOR Flash存储 ：支持32位Data

▼ **基本接口** 支持微机原理、SOC或微处理器简单应用

4×5+1矩阵按键、16位滑动开关、16位LED、8位七段数码管

▼ **标准接口** 支持基本计算机系统实现

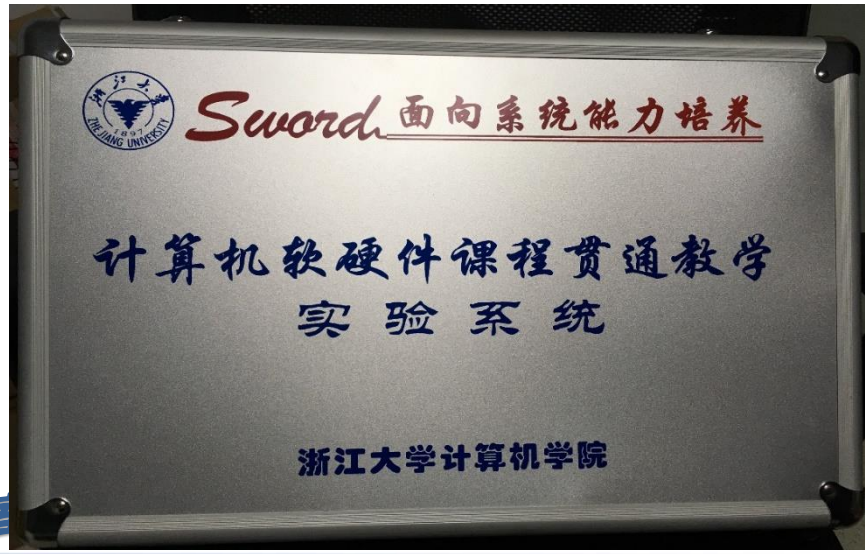
12位VGA接口（RGB656）、USB-HID（键盘）

▼ **通讯接口** 支持数据传输、调试和网络

UART接口、 10M/100M/1000M以太网、 SFP光纤接口

▼ **扩展接口** 支持外存、多媒体和个性化设备

MicroSD(TF）、PMOD、 HDMI、Arduino



Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

实验任务：选修

- **基本要求**
  - 增加SCPU中断功能
    - 兼容SCPU数据通路
    - 增加中断通路
    - 增加中断控制
  - 固定中断向量(ARM)
    - 非法指令中断
    - 外部中断
  - 此实验在兼容Exp07基础上完成
- **高级要求**
  - 支持CSR中断相关寄存器
  - 支持CSR及中断相关指令
  - 半兼容模式
    - 统一入口地址：00000004

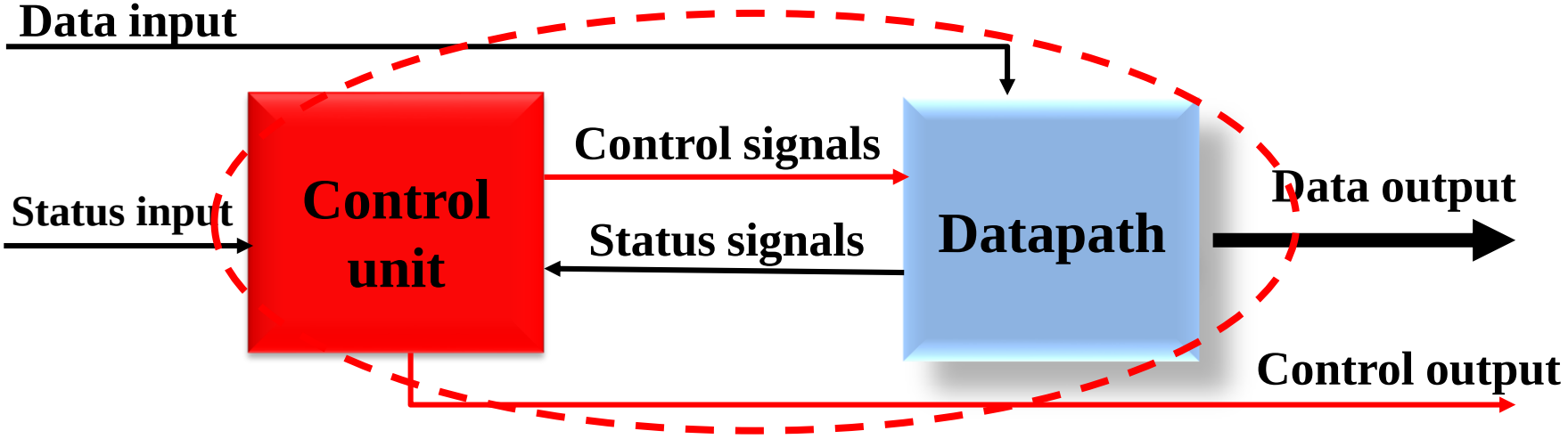
Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

CPU organization

Digital circuit

- General circuits that controls logical event with logical gates --Hardware



Computer organization

- Special circuits that processes logical action with instructions -Software

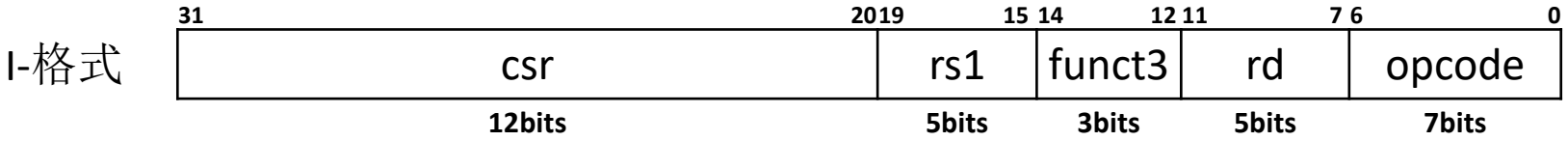
RISC-V中断结构

- **Control and Status Registers CSR**
  - RISC-V特权处理的辅助部件
    - 管理特权
      - Machine MODE具有最高特权，是所有RISC-V都必须具有的
    - 处理中断异常
    - 存储器管理
    - 系统配置
- **中断相关的常用CSR寄存器**

CSR	Privilege	Abbr.	Name	Description
0x300	MRW	mstatus	Machine STATUS register	MIE、MPIE域标记中断全局使能
0x304	MRW	mie	Machine Interrupt Enable register	控制不同类型中断的局部使能
0x305	MRW	mtvec	Machine trap-handler base address	进入异常服务程序基地址
0x341	MRW	mepc	Machine exception program counter	异常断点PC地址
0x342	MRW	mcause	Machine trap cause register	处理器异常原因
0x343	MRW	mtval	Machine Trap Value register	处理器异常值地址或指令
0x344	MRW	mip	Machine interrupt pending	处理器中断等待处理

CSR传输设置指令

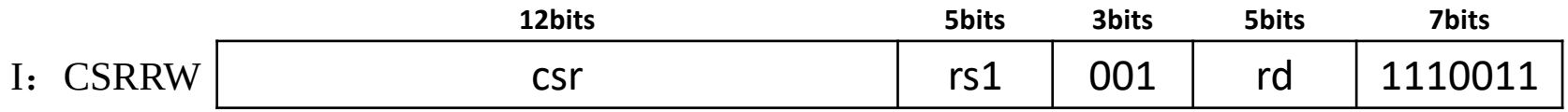
CSR指令格式



12位独立寻址空间支持4096个CSR寄存器

CSRRW -Control and Status Register Read and Write

- Atomic Read and Write CSR
- instruction atomically swaps values in the CSRs and integer registers.
- csrrw rd, csr, rs1
  - ***if*** rd! =x0 then x[rd] ← x[crs]; x[csr] ← x[rs1]



- 伪指令：rd=0，set crs： csrw csr, rs1

CSR传输设置指令

CSRRS -Control and Status Register Read and Set

- Atomic Read and Set Bits in CSR
- instruction reads the value of the CSR, and writes it to integer register rd
- csrrs rd, csr, rs1
  - x[rd] ← x[crs]; ***if*** rs1! =x0 then x[csr] ← {x[rs1] | x[csr]};

I： CSRRS	csr	rs1	010	rd	1110011

- 伪指令 rd =0， set bits in CSR： csrs csr, rs1

CSRRC -Control and Status Register Read and Clear

- Atomic Read and Clear Bits in CSR
- instruction reads the value of the CSR, and writes it to integer register rd
- csrrc rd, csr, rs1
  - x[rd] ← x[crs]; ***if*** rs1! =x0 then x[csr] ← {~x[rs1] **&** x[csr]};

I： CSRRC	csr	rs1	011	rd	1110011

- 伪指令： rs1=0， Clear crs： csrc csr, rs1

CSR立即数传输设置指令

- uimm[4:0] zero-extending a 5-bit unsigned immediate field encoded in the rs1 field instead of a value from an integer register rs1
- **csrrwi rd, csr, uimm[4:0]**
  - Control and Status Register Read and Write Immediate
    - Write CSR伪指令： csrwi csr, uimm[4:0]
- **csrrsi rd, csr, uimm[4:0]**
  - Control and Status Register Set Immediate
    - Set bits CSR伪指令：csrsi csr, uimm[4:0]
- **csrrci rd, csr, uimm[4:0]**
  - Control and Status Register Read and Clear Immediate
    - Clear bit crs伪指令：csrci csr, uimm[4:0]

	31 … 20	19 … 15	14 … 12	11 … 7	6 … 0
I：CSRRWI	csr	uimm	101	rd	1110011
I： CSRRSI	csr	uimm	110	rd	1110011
I： CSRRCI	csr	uimm	111	rd	1110011

Interrupts Instruction

- **Exception return**
  - MRET
    - PC ← MEPC；(MStatus寄存器有变化)

	31–25	24–20	19–15	14–12	11–7	6–0
R：MRET	0011000	00010	00000	000	00000	1110011
R：SRET	0001000	00010	00000	000	00000	1110011
R：wfi	0001000	00101	00000	000	00000	1110011

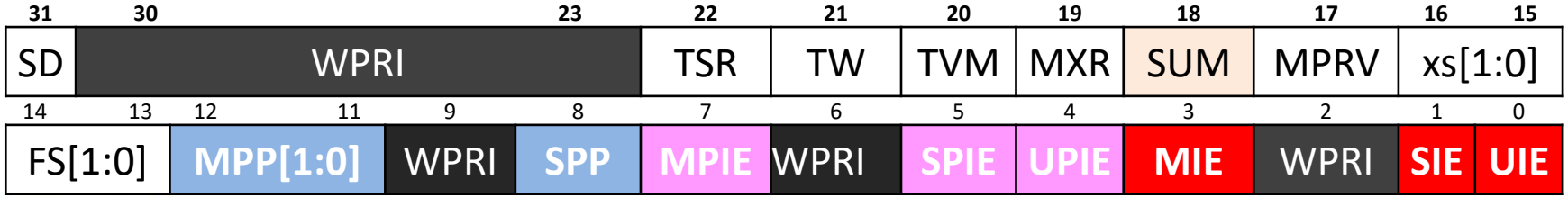
- **Environment call**
  - ecall
    - MEPC ← ecall指令本身的PC值
- **Breakpoint**
  - ebreak
    - MEPC ← ebreak指令本身的PC值

	31–25	24–20	19–15	14–12	11–7	6–0
I：ecall	0000000	00000	00000	000	00000	1110011
I：ebreak	0000000	00001	00000	000	00000	1110011

Interrupt Registers： mstatus(*0x300*)

- Machine STATUS register
  - These bits are primarily used to guarantee atomicity with respect to interrupt handlers in the current privilege mode.

bit	Name	Attributes	Description
0	UIE	RW	User interrupt enable
1	SIE	RW	Supervisor interrupt enable
3	MIE	RW	Machine interrupt enable
4	UPIE	RW	previous user-level interrupts enable
5	SPIE	RW	Previous Supervisor interrupt-enable
7	MPIE	RW	Previous Machine interrupt enable
8	SPP	RW	Supervisor Previous Privilege mode
12:11	MPP	RW	Machine Previous Privilege mode
……			
31:23,9,6,2	WPRI		Reserved



Interrupt Registers： mcause (*0x342*)

Machine Cause Register (mcause)

- When a trap is taken mcause register is written with a code indicating the event that caused the Exception/Interruption.

- Exception Code与mtvec的向量模式相对应
  - 在异步中断时，不同的模式会跳转到不同的入口
- The Exception Code is a WLRL
  - Write/Read Only Legal Values
  - 如果写入值不合法可以引发非法指令异常

Exception Code mcause (*0x342*)

INT	E Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	Reserved for future standard use
1	**3**	**Machine software interrupt**
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6	Reserved for future standard use
1	7	Machine timer interrupt
1	8	User external interrupt
1	9	Supervisor external interrupt
1	10	Reserved for future standard use
1	**11**	**Machine external interrupt**
1	12-15	Reserved for future standard use
1	≥16	Reserved for platform use
0	16-23	Reserved for future standard use
0	32-47	Reserved for future standard use
0	14/≥64	Reserved for future standard use

INT	E Code	Description
0	0	Instruction address misaligned
0	1	Instruction access fault
0	**2**	**Illegal instruction**
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	Reserved
0	11	**Environment call from M-mode**
0	12	Instruction page fault
0	13	Load page fault
0	15	Store/AMO page fault
0	24-31	Reserved for custom use
0	48-63	Reserved for custom use

Interrupt Registers： mtvec (*0x305*)

Machine Trap-Vector Base-Address Register

- The mtvec register holds trap vector configuration, consisting of a vector base address (BASE) and a vector mode (MODE)

- The value in the BASE field must always be aligned on a 4-byte boundary, and the MODE setting may impose additional alignment constraints on the value in the BASE field.

MODE Value	Name	Description
0	Direct(查询)	All exceptions set pc to BASE
1	Vectored(向量)	Asynchronous interrupts set pc to BASE+4×cause
≥2	--	Reserved

BASE+ 4×Exception Code

Interrupt Registers： mepc (*0x341*)

Machine Exception Program Counter

- mepc is a WARL register that must be able to hold all valid physical and virtual addresses.
  - When a trap is taken into M-mode, mepc is written with the address of the instruction that was interrupted or exception.

- The low bit of mepc (mepc[0]) is always zero.
  - On implementations that support only IALIGN=32, the two low bits (mepc[1:0]) are always zero.
- Exception： `mepc ← pc`
- Interrupted： `mepc ← pc + 4`

RISC-V中断响应(Only M-MODE)

- 初始化
  - 设置mstatus：关中断 MIE=0
  - 系统初始化
  - 设置mstatus：开中断 MIE=1
- 中断响应
  - 硬件保存断点：MEPC ←PC/PC+4
  - 硬件修改PC：PC ← mtvec(向量 、硬件关中断)
  - 中断服务：保护寄存器、开中断*、服务、恢复寄存器
  - 开中断*
  - 返回：mret （硬件修改Cause和Status寄存器）

RISC-V中断处理--进入异常

- **RISC-V处理器检测到异常，开始进行异常处理：**
  - 停止执行当前的程序流，转而从CSR寄存器mtvec定义的PC地址开始执行；
  - 更新机器模式异常原因寄存器：mcause
  - 更新机器模式中断使能寄存器：mie
  - 更新机器模式异常PC寄存器：mepc
  - 更新机器模式状态寄存器：mstatus
  - 更新机器模式异常值寄存器：mtval

RISC-V中断结构--退出异常

- 异常程序处理完成后，需要从异常服务程序中退出，并返回主程序
  - RISCV中定义了一组退出指令MRET，SRET，和URET
  - 机器模式对应MRET。
- 机器模式下退出异常(MRET)
  - 程序流转而从csr寄存器mepc定义的pc地址开始执行
  - 同时硬件更新csr寄存器机器模式状态寄存器mstatus
    - 寄存器MIE域被更新为当前MPIE的值：mie ← mpie
    - MPIE 域的值则更新为1： MPIE ← 1

中断数据通路和控制器

- 数据通路
  - 至少需要增加CSR：
    - mepc(341)、mstatus(300)、mtvec(305)、mcause(342)
  - 必须增加MEPC寄存器及MEPC←PC/PC+4通路
  - 增加CSRRW/CSRRS 和mret指令通道
- 控制器
  - 增加中断检测电路
  - 增加异常检测电路
  - 中断响应控制
  - CSR传输控制

典型处理器中断结构

- **Intel x86中断结构**
  - 中断向量：000～3FF，占内存最底1KB空间
    - 每个向量由二个16位生成20位中断地址
    - 共256个中断向量，向量编号n=0～255
    - 分硬中断和软中断，响应过程类同，触发方式不同
    - 硬中断响应由控制芯片8259产生中断号n(接口原理课深入学习)
- **ARM中断结构**
  - 固定向量方式(嵌入式课程深入学习)

异常类型	偏移地址(低)	偏移地址(高)	
复 位	00000000	FFFF0000	
未定义指令	00000004	FFFF0004	
软中断	00000008	FFFF0008	
预取指令终	0000000C	FFFF000C	
数据终止	00000010	FFFF0010	
保留	00000014	FFFF0014	
中断请求(IRQ)	00000018	FFFF0018	
快速中断请求(FIQ)	0000001C	FFFF001C	

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

中断设计内容

- **确定中断向量模式**
  - 中断向量：中断入口寻址方法
    - 固定：直接给出中断入口地址： mtvec(BASE)
    - 动态：计算获得中断入口地址： BASE + 4×Exception Code
- **数据通路**
  - 需要**增加**CRS的Cause和Status寄存器
  - 必须增加EPC寄存器及EPC←PC/PC+4通路
  - 增加CSRRW/CSRRS和mret指令通道
- **控制器**
  - 增加中断检测电路
  - 增加异常检测电路
  - 增加中断响应控制
  - 增加CSR传输控制

简化中断设计：ARM模式

简化中断设计

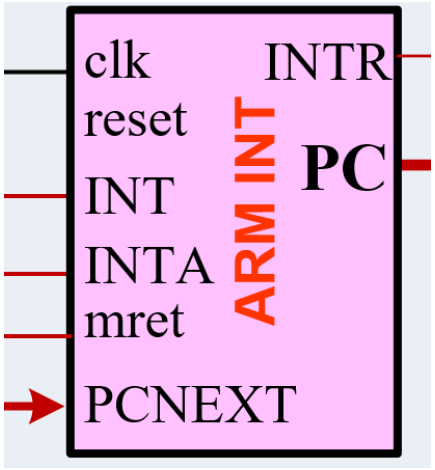
- 采用ARM中断向量(不兼容RISC-V)
  - 实现非法指令异常和外中断
  - 设计EPC
- ARM中断向量(BASE = 0x0000004，其余兼容RISC-V*)
  - 仅M-Mode中断寄存器(MCause、Mstatus、MIE、MIP、MEPC和MTVEC*)
  - 设计mret、CSRRW (csrw rd, csr, rs1)和ecall指令

ARM中断向量表

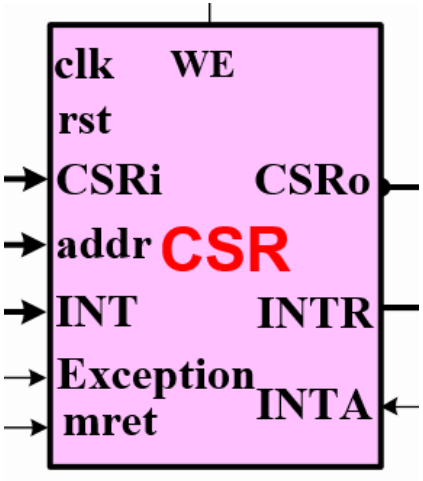
向量地址	ARM异常名称	ARM系统工作模式	本课程定义
0x0000000	复位	超级用户Svc	复位(M-MODE)
0x0000004	未定义指令终止	未定义指令终止Und	非法指令异常
0x0000008	软中断（SWI）	超级用户Svc	ECALL
0x000000c	Prefetch abort	指令预取终止Abt	Int外部中断（硬件）
0x0000010	Data abort	数据访问终止Abt	Reserved自定义
0x0000014	Reserved	Reserved	Reserved自定义
0x0000018	IRQ	外部中断模式IRQ	Reserved自定义
0x000001C	FIQ	快速中断模式FIQ	Reserved自定义

DataPath扩展中断通路

- **DataPath修改**
  - 修改PC模块增加(ARM模式)
    - CPU复位时IE=0，EPC=PC=ox00000000
    - IE=中断使能**(重要)**
    - EPC寄存器，INT触发PC转向中断地址
      - 相当于硬件触发Jal，用eret返回
    - 增加控制信号INT、RFE/eret
      - INT宽度根据扩展的外中断数量设定
  - 修改PC模块增加(RISC-V模式)*
    - CSR简化模块
      - MEPC、MCause和MStatus寄存器等
      - 增加CP0数据通道
        - MEPC、MCause和Status通道
      - 增加控制信号CSR_Write
    - 修改PC通道



ARM中断模块



RV32中断模块

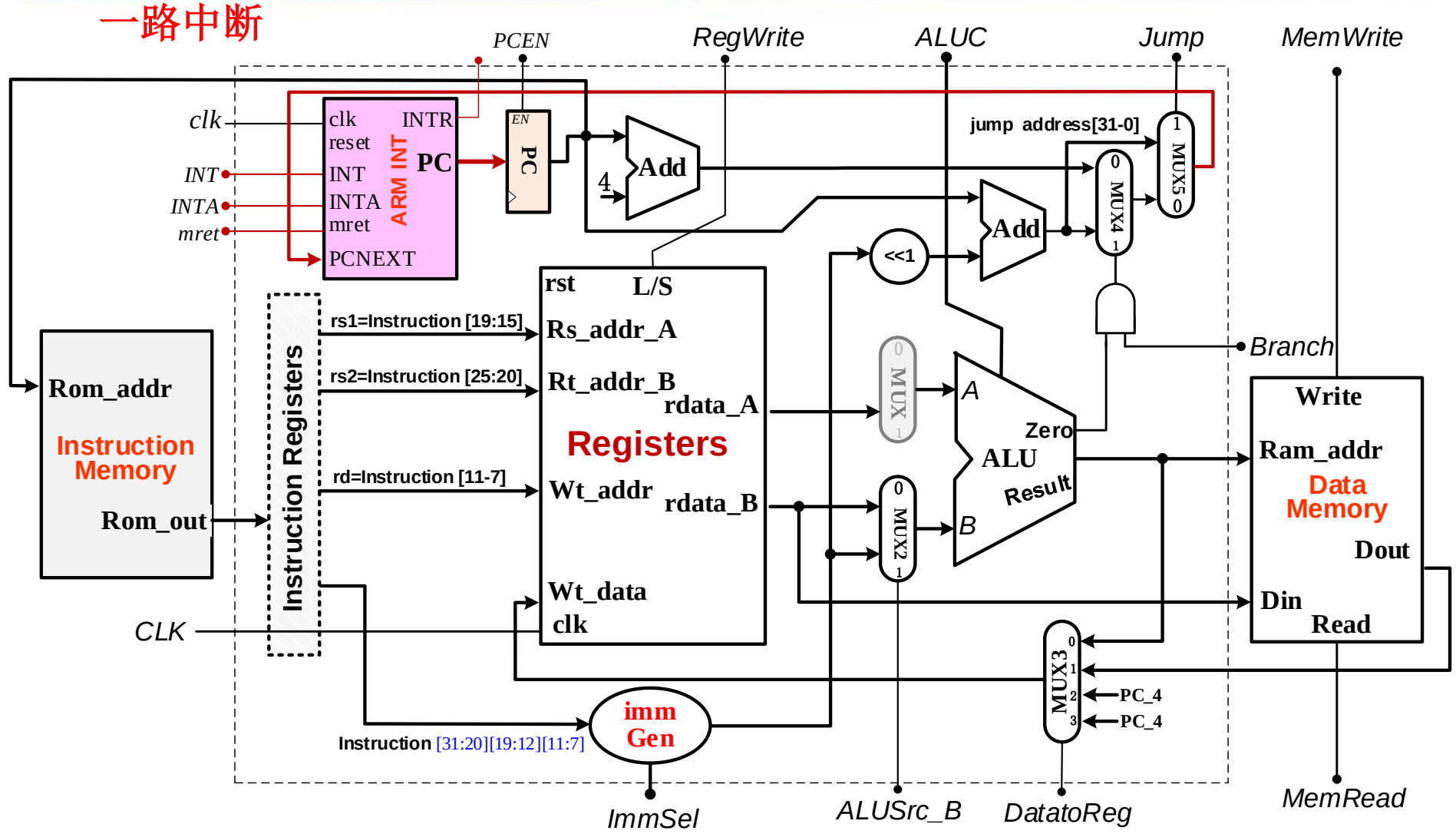
注意：INT是电平信号，不要重复响应

控制器扩展中断译码

- **控制器修改**
  - ARM模式(简单)
    - 仅增加mret指令
    - 中断请求信号触发PC转向，在Datapath模块中修改
  - RISC-V模式(可独立模块)*
    - 至少扩展mret、CSRRW指令译码
    - 增加Wt_Write通道选择控制
    - 增加CSR_Write
    - 增加控制信号mret、ecall、CSR_Write
- **中断调试**
  - 首先时序仿真
  - 物理验证
    - 用BTN[0]触发调试：静态或低速
    - 用计数器counter0_OUT调试：动态或高速

      注意动态测试时死锁!!!

ARM中断模式：Single cycle DataPath



ARM interrupt description

- 中断触发检测与服务锁存

检测与锁存

```
//interrupt Trigger
      assign INTR = int_act;
      assign int_clr = reset| ~int_act;          //clear interrupt Request

      always @(posedge clk)
      INT_get <= {INT_get[0],INT};               //Interrupt Sampling

      always @(posedge clk)begin                 //interrupt Request
         if(INT_get==2'b01)int_req_r<=1;         //set interrupt Request(相当于MEIP)
         else if(int_clr)int_req_r<=0;           //clear interrupt Request
      end
```

- 断点保护、中断开、并与返回

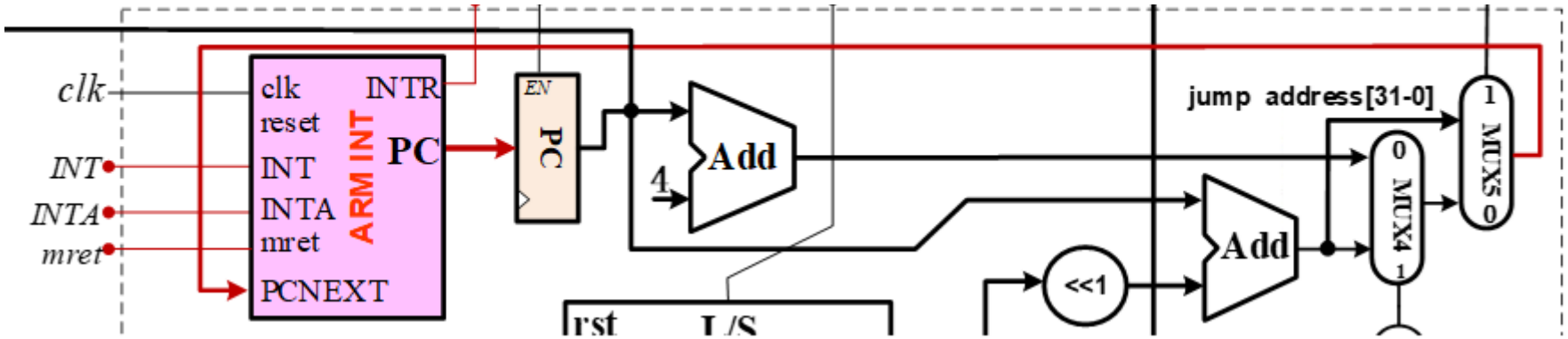
```
always @(posedge clk or posedge reset ) begin
      if (reset)begin EPC     <= 0;            //EPC=32'h00000000;
                      int_act <= 0;
                      int_en  <= 1;            //相当于MIE
      end
      else if(int_req_r & int_en)begin         //int_req_r: interrupt Request reg
                  int_act <= 1;                //interrupt Service set
                  int_en  <= 0;                //interrupt disable
           end
           else begin if(INTA & int_act)
                      begin int_act<=0;
                        EPC <= pc_next;        //interrupt return PC
                        end
                      if(mret) int_en<=1;      //interrupt enable if pc<=EPC;
                  end
  end
```

中断通路模块：PC通路

- PC输出通路

仅实现一路中断，请同学们扩展

```
//PC Out
    always @* begin
        if (reset==1)pc <= 32'h00000000;
        else if(INTA)
                pc <= 32'h00000004;        //interrupt Vector    & int_en
            else if(mret)pc <= EPC;        //interrupt return
                else pc <= pc_next;        //next instruction
    end
```

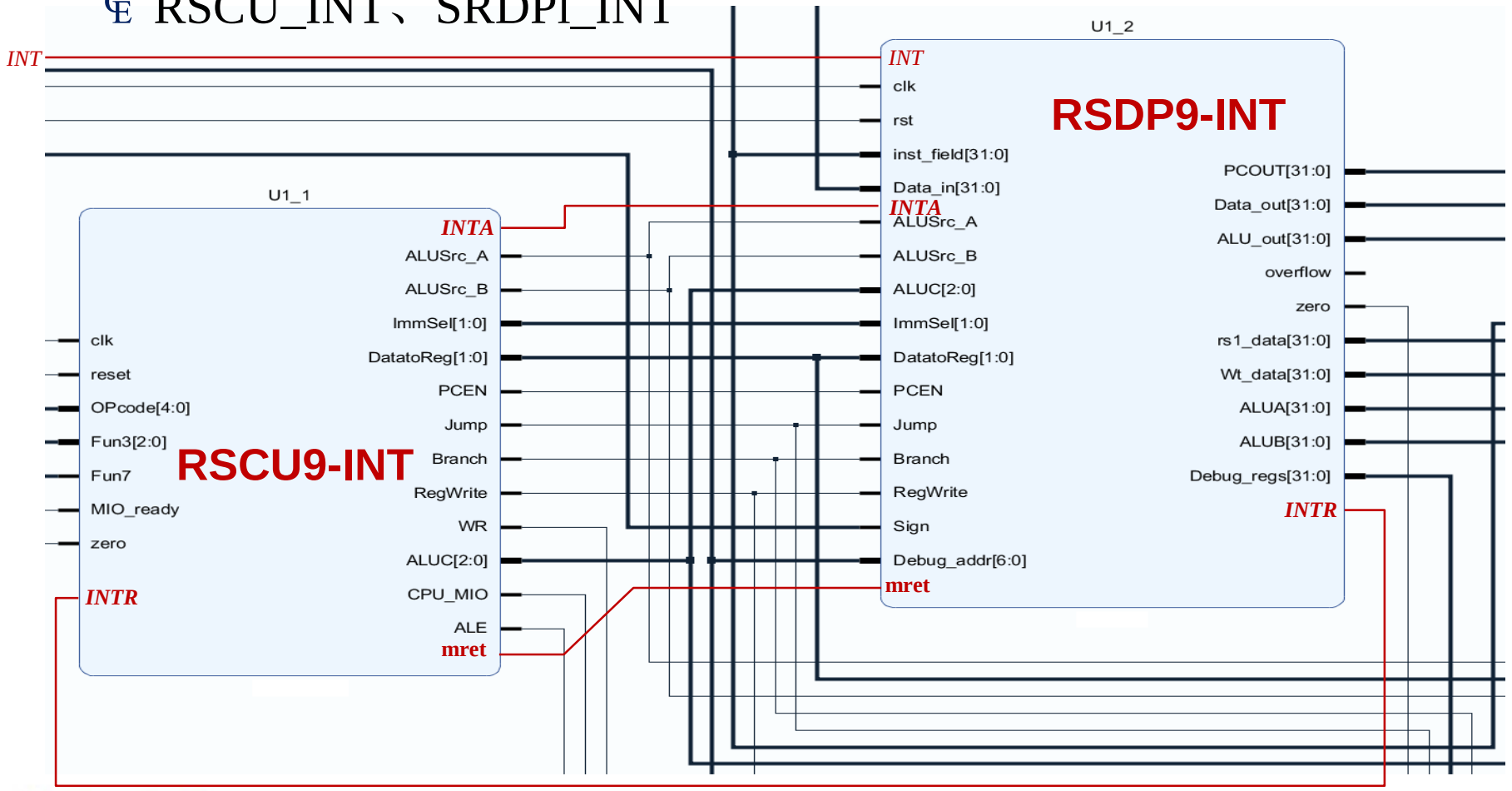


- RSCU_INT ?

INTA ⟷ INTR

增加中断后的CPU模块

- 注意增加信号和模块互连
  - RSCU_INT、SRDPl_INT



修改功能测试程序：判断子代码

```
loop1:
      lw a1, 0x0(s1)          #读GPIO端口F0000000状态，同前。
      add a1, a1, a1;
      add a1, a1, a1;         #左移2位将SW与LED对齐
      sw a1, 0x0(s1);         #再将新$a1:r5写到GPIO端口F0000000,写到LED
      lw a1, 0x0(s1);         #再读GPIO端口F0000000状态
      and s8, a1, t0;         #与80000000相与，即：取最高位=out0,屏蔽其余位
      add s6, s6, t1;         #程序计数延时(加1)
      #beq   s8, t0, C_init;  #硬件计数。out0=1,Counter通道0溢出,转C_init
      #beq s6, zero, C_init;  #若程序计数$t5:r13=0,转C_init
l_next:                       # 延时未到，继续：判断7段码显示模式：SW[4:3]
      lw a1, 0x0(s1) ;        #再读GPIO端口F0000000开关SW
      add s7, a4, a4;         #因x14=4, 故s7：x23=00000008
      add s9, s7, s7;         #s9：x25=00000010
      add s7, s7, s9;         #s7：x23=00000018(00011000)：11对应SWO[4:3]
      and s8, a1, s7;         #取SW[4:3]：屏蔽其余位送x24
      beq s8, zero, L00;      #SW[4:3]=00, L00：7段显示"点"循环移位，SW0=0
      beq s8, s7,    L11;     #SW[4:3]=11，L11：显示七段图形，SW0=0
      add s7, a4, a4 ;        #$s2:r18=8
      beq s8, s7,    L01 ;    #SW[4:3]=01, L01：显示内存预置16进制值
L10：……                      #SW[4:3]=10，L10显示x21(即时值+1)，SW0=1(用户扩展：)
```

SWO状态循环显示

硬件/软件/中断三种定时选择

注释掉

功能判断

修改定时子代码：中断返回

```
C_init:                          # 延时结束，修改显示值和定时/延时初始化
        lw s6, 0x14(zero) ;      # 取程序计数延时初始化常数
        add a5, a5, a5;          # a5左移，x15=xxxxxxx0，七段图形点左移
        or a5, a5, t1;           # a5:x15末位置1，消除七段显示器右上角点，不显示
        add s3, s3, a4;          # x14=00000004，LED图形访存地址+4
        and s3, s3, s0;          # 和3F相与，x19=000000xx，屏蔽高位，简单截取低位地址(6位)
        add s5, s5, t1 ;         # x21+1
        beq s5, a3, L6 ;         #若x21=ffffffff，重置x21=5
        j L7;
L6:
        add s5, zero, a4 ;       #x21=4
        add s5, s5, t1 ;         #重置x21=5
L7:
        lw a1, 0x0(s1) ;         #读GPIO端口F0000000状态
        add s8, a1, a1;
        add s8, s8, s8 ;         # 左移2位将SW与LED对齐，同时D1D0置00，选择计数器通道0
        sw s8, 0x0(s1);          # x24输出到GPIO端口F0000000，计数器通道counter_set=00端口
                                 不变、LED=SW：  {GPIOf0[15:2], LED, GPIOf0[1:0]/counter_set}
        sw a2, 0x4(s1)           # 计数器端口:F0000004，送计数常数x12=F8000000
        mret;                    #本处直接跳转，若中断或子程序则调用则返回
```

注：硬件计数与计数中断时要判断硬件计数溢出已经消除，否则会造成多次进入。

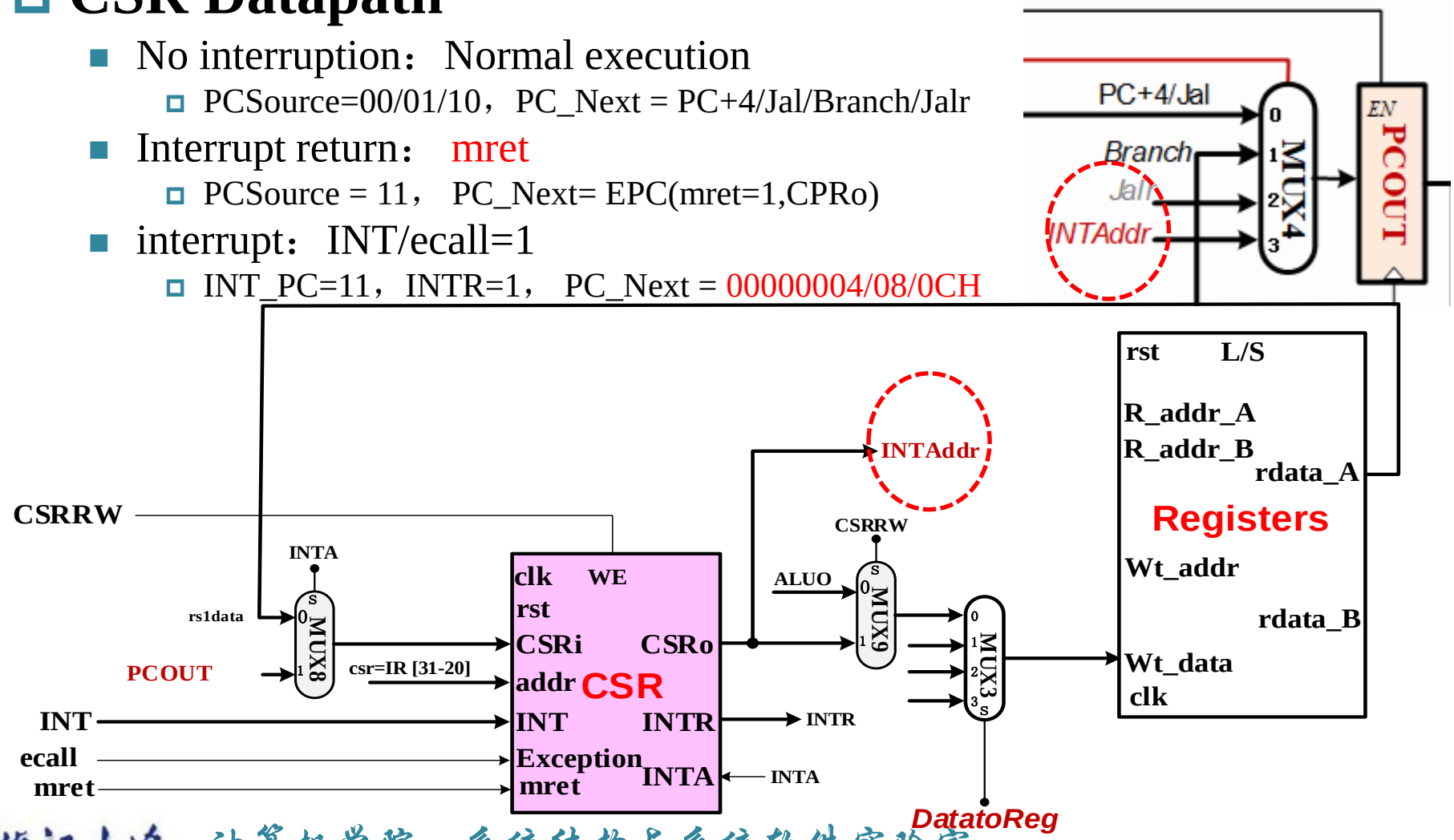
注意动态测试时死锁!!!

兼容RISC-V M-MODE 中断设计

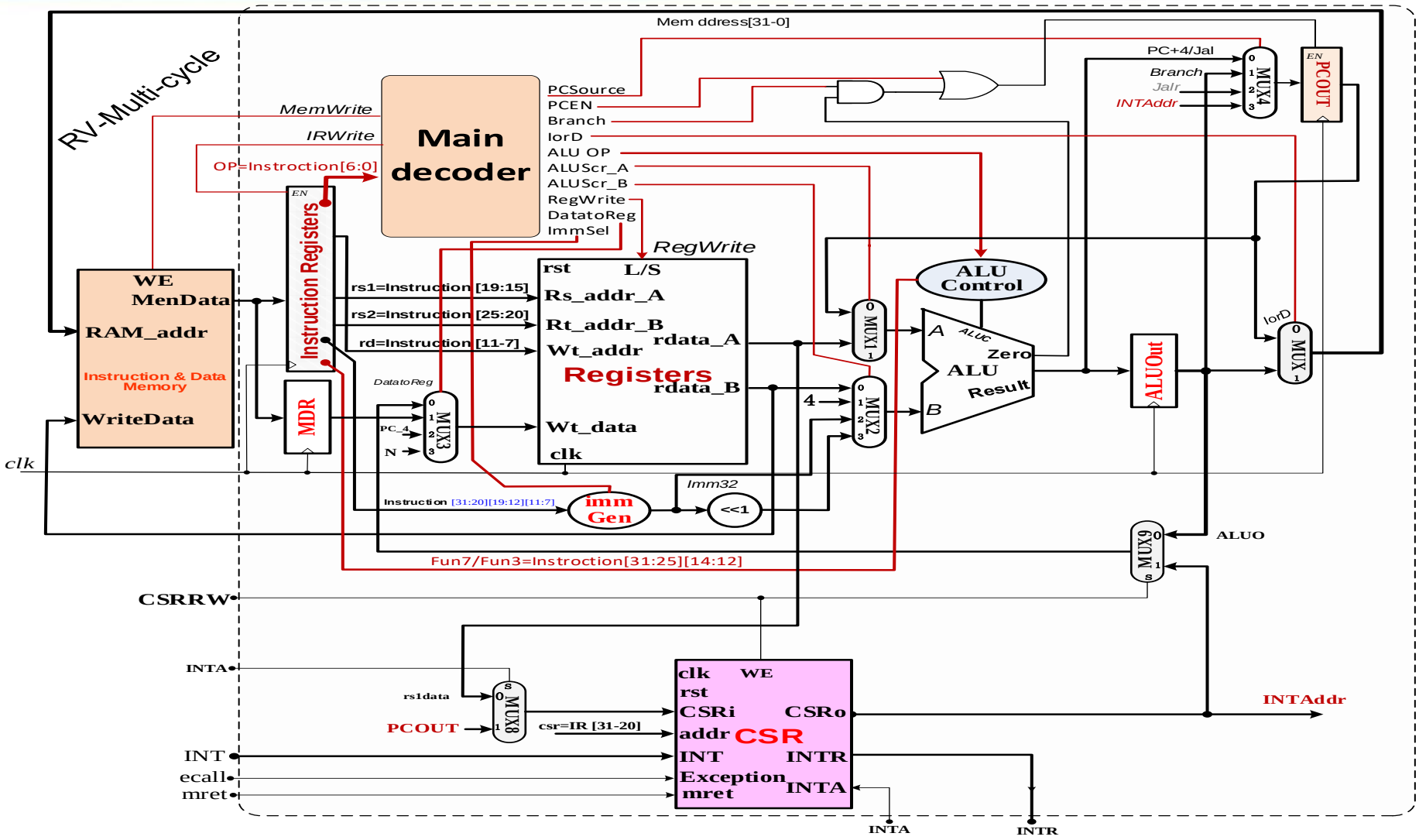
BASE = 0x0000004

CSR Datapath

- No interruption： Normal execution
  - PCSource=00/01/10， PC_Next = PC+4/Jal/Branch/Jalr
- Interrupt return： mret
  - PCSource = 11， PC_Next= EPC(mret=1,CPRo)
- interrupt： INT/ecall=1
  - INT_PC=11， INTR=1， PC_Next = 00000004/08/0CH



Multi-cycle Interruption Implementation



CSR Interrupt Description

```
1  module      CP0(input  clk,rst,
2                  input  INTA,
3                  input  [4:0] addr,
4                  input  [31:0]CPRi,
5                  input  WE,
6                  input  [7:0]INT,
7                  input  Exception,
8                  input  eret,
9                  output [31:0] CPRo,
10                 output [31:0] Cause,
11                 output [31:0] Status,
12                 output INTR );
13 reg [31:0] CP0 [0:3];                    // CP0.0-CP0.3
14 reg [7:0]Trig;
15 reg [1:0] req_get;
16 reg int_clr,int_act;
17 integer i;
18
19    wire [4:0] Exc_code = {1'b0,Exception,3'b000};
20    assign EPC   = CP0[2];
21    assign Cause    = CP0[1];
22    assign Status   = CP0[0];
23    assign int_en   = Status[0];
24    assign INTR = int_act;
25    wire [7:0] INT_MK    = INT & Status[15:8];          //TInterrupt mask
26    assign int_req = | {INT_MK, Exception };
27    always @(posedge clk)
28       req_get <= {req_get[0], int_req};               //外部中断采样
```

这是MIPS中断CP0简化描述。CSR相当于CP0的变形，请同学根据CRS相关中断寄存器定义修改CP0实现RISC-V m-mode简化中断。

CP0部分修改为相应的CSR及功能定义

ExcCode：sys=08=5’b01000

CSR Interrupt Description-1

```
29  //interrupt Trigger
30     always @(posedge clk)
31        if(rst)
32           int_act <= 0;
33        else if(int_en && (req_get=2'b01))
34                int_act <= 1;                                  //*interrupt Service
35             else if(INTA & int_act)                           // ACK interrupt Service
36                 int_act<=0;                                   // clear interrupt Request
37
38  // CP0 Register Access
39     assign CPRo =(~eret) ? CP0[{addr[1:0]}] : EPC ;           // read
40
41     always @(posedge clk or posedge rst) begin
42           if (rst==1) begin                                   // reset
43           int_clr <=0 ;
44           CP0[0] <= 32'h0000FF00;
45           for (i=1; i<4; i=i+1)CP0[i] <= 0;                   //CP0 clear;
46        end
47        else begin
48             if ((addr[4:2] == 3'b011) &&(WE == 1))           // write
49                 CP0[{addr[1:0]}] <= CPRi;
50              else begin
51                 if(INTA | Exception)begin
52                 CP0[2]  <= CPRi;                          //Save interrupt return PC
53                 CP0[1]  <= {Cause[31:16],INT_MK,1'b0,Exc_code,2'b00};
54                 CP0[0]  <= {Status[31:1], 1'b0};                          修改
55                 end
56                 if(eret) CP0[0]  <= {Status[31:1], 1'b1};    //restore interrupt enable if pc<=EPC
57              end
58        end
59     end
60  endmodule
```

思考题

- 如何设计中断静态测试(BTN0)？
- 如何扩展单周期中断通路模块实现多路中断？
- 如何扩展流水线中断？
- 流水线出现多个中断如何处理？


END