疑问1、data\_load\_event     // data request is in the special event range elw( event load) ELW\_EXE状态

2、Dscr\_t结构体为[31:1] 而被赋值的csr\_rdata\_int为[31:0]

3、pccr : performance counters counter register

Pcmr: mode register, controls saturation and global enable

Pcer: selected counter input

4、debug\_single\_step\_o = dcsr\_q.step;(csr第917行,但dcsr\_q.step未注明) ——>controller内554-580行？

5、perf\_pipeline\_stall\_o？作用

注释：Core中对中断判断：

is\_interrupt = (pc\_mux\_id == PC\_EXCEPTION) && (exc\_pc\_mux\_id == EXC\_PC\_IRQ)

其中logic [2:0]        pc\_mux\_id;     // Mux selector for next PC

  logic [2:0]        exc\_pc\_mux\_id; // Mux selector for exception PC

int.controller.sv 用于处理中断状态：分IDLE，PENDING，DONE，当处于PENDING时将irq\_req\_ctrl信号置高并传递给controller

controller内的有限状态机：

**WAIT\_SLEEP：**等待一个周期后进入SLEEP状态

**SLEEP：**停留在此状态表示正在排空流水线，直到没有irq( wfi == nop) ,或者处于debug mode/single step mode(单步调试模式) 离开进入decode状态

**FIRST\_FETCH：**主要用于处理取指阶段中遇到debug\_req和irq\_req，分别转入DEBUG\_TAKEN\_IF和IRQ\_TAKEN\_IF状态，否则进入DECODE状态

**DECODE：**若发生EX段有branch、data\_err或fetch\_fail，将is\_decoding置零(仅用于csr中不计入指令数counter)，第一者设置pc\_mux于pc\_branch；

第二者通过csr\_save\_ex置高表示要保存ex段的pc值，并传递data\_err\_ack握手信号告知pmp，而后进入FLUSH\_WB状态；

第三者通过csr\_save\_if置高表示要保存if段的pc值，进入FLUSH\_WB状态；

其余情况下is\_decoding置高，判断有无irq\_req/debug\_req（处理同FIRST\_FETCH），若都为低，则设置exc\_kill传入int.controller的ctrl.kill，退出中断，再判断illegal\_insn，若为高则表示要保存id的pc值，进入FLUSH\_EX状态；

否则依次判断jump\_in(等待jump指令地址计算), ebrk\_insn（接着判断是否是debug模式，若是，进入DEBUG\_FLUSH，否则csr\_save\_id置高要保存id的pc值并进入FLUSH\_EX）, pipe\_flush(正常排空流水线，进入FLUSH\_EX), {ecall\_insn(要求保存id段pc), fencei\_insn, mret\_insn || uret\_insn || dret\_insn（退出三种模式的指令）}（进入FLUSH\_EX）, csr\_status（id\_ready则进入FLUSH\_EX，否则停留在DECODE）, data\_load\_event（id\_ready则进入ELW\_EXE，否则停留在DECODE）

若刚进入debug的单步调试模式（debug\_single\_step\_i & ~debug\_mode\_q），若id\_ready(有指令刚被执行)，若是illegal, ebrk, mret, uret指令则进入FLUSH\_EX状态；若是branch则进入DBG\_WAIT\_BRANCH状态，否则默认进入DBG\_FLUSH状态

**FLUSH\_EX：**若data\_err || ex\_valid 则进入FLUSH\_WB。data\_err时通过data\_err\_ack与pmp握手，要求保存id段pc值，并把illegal置零

**IRQ\_FLUSH：**data\_err则要求保存id段pc值并进入FLUSH\_EX；否则若有irq则进入IRQ\_TAKEN\_ID，否则进入DECODE

（// we can go back to decode in case the IRQ is not taken (no ELW REPLAY)）

**ELW\_EXE：**special event load ?

**IRQ\_TAKEN\_ID：**中断执行，irq\_ack, exc\_ack置高，pc\_mux设为异常，异常指针（exc\_pc\_mux）设为中断（指针类型仅有异常无中断，异常指针类型包括中断，异常和debug），要求保存id段pc值，传递exc\_cause至if段的exc\_vec\_pc\_mux（selects ISR address for vectorized interrupt lines），转入DECODE状态

**IRQ\_TAKEN\_IF：**同上，区别仅在于传给csr\_save是id还是if段的（即要求保存if段pc值）

**FLUSH\_WB：**默认转入DECODE状态，依次：

1、判断data\_err, 并判断data\_we\_ex，决定异常类型为LOAD\_FAULT还是STORE\_FAULT；

2、判断is\_fetch\_fail 异常类型为INSTR\_FAULT

3、判断illegal\_insn，并判断是否刚进入debug单步调试，若是则转入DBG\_TAKEN\_IF状态

4、判断ebrk\_insn，并判断是否刚进入debug单步调试，若是则转入DBG\_TAKEN\_IF状态

5、判断ecall\_insn，异常类型为ECALL\_MMODE，并判断是否刚进入debug单步调试，若是则转入DBG\_TAKEN\_IF状态

6、判断mret\_insn/uret\_insn/dret\_insn，转入XRET\_JUMP

7、判断pipe\_flush，转入WAIT\_SLEEP

8、判断fencei\_insn，pc\_mux置为PC\_FENCEI(if内pc+4)（注此时默认转入DECODE）

**XRET\_JUMP：**mret/uret/dret(退出三种模式)后pc恢复到之前保存的值，若刚进入debug单步调试模式，则转入DEBUG\_TAKEN\_IF

**DEBUG\_WAIT\_BRANCH：**判断branch\_taken\_ex，若是则把pc\_mux置为BRANCH，转入DEBUG\_FLUSH状态（？未写在if判断之内）

**DEBUG\_TAKEN\_ID/DEBUG\_TAKEN\_IF：**在ID/IF段进入debug模式，并要求保存对应pc值

**DEBUG\_FLUSH：**有debug\_req时先进入此状态，判断有无data\_err，若有，先进入FLUSH\_EX状态在直接转入DEBUG\_TAKEN\_IF，否则若是debug单步调试请求也进入DEBUG\_TAKEN\_IF，否则进入DEBUG\_TAKEN\_ID。

中断类型：1、INSTR\_FAULT

2、ILLEGAL\_INSN

3、BREAKPOINT

4、LOAD\_FAULT

5、STORE\_FAULT

6、ECALL\_UMODE

7、ECALL\_MMODE

仲裁：优先级：PRIV\_LVL 分M/H/S/U四个由高到低

（irq\_sec\_i信号由核外传入，什么作用？）

（感觉优先级比较在csr的611行左右开始，具体没看明白）

如何提交至现有指令流：有中断请求时先halt if,id段。IRQ\_TAKEN\_ID/IF等状态给pc赋值跳转

如何保存和恢复现场：

保存：core.sv中第1051-1057行，把forwarding到id段的alu的值存入csr的mepc/uepc/depc结构体的对应的地址、数据和操作数的数值中，pc值则由controller置高要求保存哪段的pc值，由if将此刻if、id、ex的pc值全传入csr后选择对应的并存入结构体。

恢复：XRET\_JUMP状态下pc\_mux被赋予对应模式下的值，经if段从csr中取出mepc/uepc/depc值

（csr\_addr/csr\_wdata /csr\_rdata在csr中不知道什么作用，感觉只有保存浮点运算相关的操作，不知道如何恢复现场）