中断与异常检测处理说明

1. **异常检测类型**

此次实验异常处理只实现6种异常：硬件复位、中断（软中断、硬中断）、syscall系统调用、无效指令、溢出、自陷指令引起的异常。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **优先级** | **异常** | **描述** | **检测阶段** | **excepttype对应位** |
| 1 | Reset | 硬件复位 |  |  |
| 7 | Interrupt | 8个中断，6个外部硬件中断，2个软件中断 | MEM | [7:0] |
| 16 | Sys | 执行了系统调用指令syscall | ID | [8] |
| Ri | 无效指令 | ID | [9] |
| Ov | 算术操作指令add、addi、sub的溢出 | EX | [10] |
| Tr | 执行了自陷指令 | EX | [11] |
| 无 | eret | 异常返回 | ID | [12] |

不同阶段进行相应的异常检测，流水线将异常信息传递下去，到MEM阶段进行统一的异常处理，以实现精确异常处理。

流水线异常信息：excepttype[31:0]与current\_inst\_addr[31:0]，前者用于记录发生异常类型，后者用于存储当前指令地址并用于精确异常处理。

异常返回处理信息：is\_in\_delayslot，用于异常返回地址确定。需要检测到分支时记录并在下一条指令信息中记录为延迟槽指令。

异常类型excepttype[31:0]编码如下：（记录时全部位或即可）

excepttype[7:0]：对应8个中断，interrupt[7]~interrupt[0]

excepttype[8]：对应是否发生系统调用syscall异常

excepttype[9]：对应是否为无效指令

excepttype[10]：对应是否溢出

excepttype[11]：对应是否为自陷指令

excepttype[12]：对应是否为返回异常

1. **不同异常检测措施**
2. 译码阶段（检测3个异常）：系统调用、无效指令、异常返回

系统调用：译码器检测是否为syscall指令即可

无效指令：译码器检测是否为未定义指令即可

异常返回：译码器检测是否为eret指令即可

1. 执行阶段（检测2个异常）：溢出异常、自陷指令

溢出异常：译码器需要针对add、addi、sub指令输出一个overflow\_detect信号并与alu部件输出的overflow部分相与，得出的值在exceptype中记录。

自陷指令：(teq、tge、tgeu、tlt、tltu、tne ; teqi、tgei、tgeiu、tlti、tltiu、tnei)

对以上指令在译码器处生成控制信号，并和alu部件输出的zero、less信号一同判断是否存在该异常，最后在相应exceptype位中记录。

1. 访存阶段（检测1个异常）：中断

在该阶段中断处理部件中进行检测，检测方式：（该部分cp0的值由于在WR阶段存在转发需要和cp0传过来的值综合判断哪个是最新的值）

7号中断：

(cp0.cause[15] & cp0.status[15] ) && cp0.status[1] == 1’b0 && cp0.status[0] == 1’b1

6号中断：

(cp0.cause[14] & cp0.status[14] ) && cp0.status[1] == 1’b0 && cp0.status[0] == 1’b1

5号中断：

(cp0.cause[13] & cp0.status[13] ) && cp0.status[1] == 1’b0 && cp0.status[0] == 1’b1

4号中断：

(cp0.cause[12] & cp0.status[12] ) && cp0.status[1] == 1’b0 && cp0.status[0] == 1’b1

3号中断：

(cp0.cause[11] & cp0.status[11] ) && cp0.status[1] == 1’b0 && cp0.status[0] == 1’b1

2号中断：

(cp0.cause[10] & cp0.status[10] ) && cp0.status[1] == 1’b0 && cp0.status[0] == 1’b1

1号中断：

(cp0.cause[9] & cp0.status[9] ) && cp0.status[1] == 1’b0 && cp0.status[0] == 1’b1

0号中断：

(cp0.cause[8] & cp0.status[8] ) && cp0.status[1] == 1’b0 && cp0.status[0] == 1’b1

若上述条件成立，则在相应的异常记录位进行记录即可。

1. **异常处理过程（MEM阶段）**

异常类型判断 –> Control unit 部件输出异常处理地址(cu\_vector)并冲刷之前指令 -> PC0修改Status.EXL、Cause.ExcCode、Cause.BD、EPC字段

三个过程均在一个周期内完成。

1、异常类型判断：

在MEM阶段设置异常类型判断部件，用于判断是否处理异常以及根据优先级输出异常类型。

不处理中断及异常情况：

1. current\_inst\_addr == 32‘h0（复位状况或刚发生异常）
2. EX/MEM.reset == 1’b1 （清除流水线）
3. EX/MEM.enable = 1’b0 （流水线暂停）

处理优先级：中断0~7 > 系统调用 > 无效指令 > 自陷异常 > 溢出异常 > 异常返回

对多个异常优先级进行判断并输出此次处理异常类型的编码。

输出异常类型编码：（优先级由上到下递减）

|  |  |  |  |
| --- | --- | --- | --- |
| **异常** | **描述** | **输出编码** | **原纪录位置** |
| 无异常 | 无异常 | 32’h0 |  |
| Interrupt0 | 0号中断 | 32’h1 | excepttype[0] |
| Interrupt1 | 1号中断 | 32’h2 | excepttype[1] |
| Interrupt2 | 2号中断 | 32’h3 | excepttype[2] |
| Interrupt3 | 3号中断 | 32’h4 | excepttype[3] |
| Interrupt4 | 4号中断 | 32’h5 | excepttype[4] |
| Interrupt5 | 5号中断 | 32’h6 | excepttype[5] |
| Interrupt6 | 6号中断 | 32’h7 | excepttype[6] |
| Interrupt7 | 7号中断 | 32’h8 | excepttype[7] |
| Sys | 系统调用异常 | 32’h9 | excepttype[8] |
| Ri | 无效指令 | 32’ha | excepttype[9] |
| Ov | 溢出异常 | 32’hb | excepttype[10] |
| Tr | 自陷异常 | 32’hc | excepttype[11] |
| eret | 异常返回 | 32’hd | excepttype[12] |

**注意：如果有异常则MEM阶段的访存指令就是错误的，不能执行，因此，需要对访存使能进行修改，先判断是否有异常如果有则不执行访存指令。**

2、Control unit部件输出异常处理地址(cu\_vector) 并冲刷之前指令

输出地址后将PC改为异常处理程序地址，冲刷掉之前流水段(IF/ID、ID/EX、EX/MEM)的指令，使其不再运行，而WR阶段指令可以继续执行。

目前不确定异常处理程序地址，因此暂时全部设为0x80000000，此处需要后期进行修改。

（无效指令在测试时，直接暂停所有流水线段寄存器以及pc）

|  |  |  |
| --- | --- | --- |
| **异常** | **描述** | **异常处理地址** |
| 无异常 | 无异常 | 无 |
| Interrupt0 | 0号中断 | 32’h80000000 |
| Interrupt1 | 1号中断 | 32’h80000000 |
| Interrupt2 | 2号中断 | 32’h80000000 |
| Interrupt3 | 3号中断 | 32’h80000000 |
| Interrupt4 | 4号中断 | 32’h80000000 |
| Interrupt5 | 5号中断 | 32’h80000000 |
| Interrupt6 | 6号中断 | 32’h80000000 |
| Interrupt7 | 7号中断 | 32’h80000000 |
| Sys | 系统调用异常 | 32’h80000000 |
| Ri | 无效指令 | 32’h80000000 |
| Ov | 溢出异常 | 32’h80000000 |
| Tr | 自陷异常 | 32’h80000000 |
| eret | 异常返回 | cp0.epc |

1. PC0修改Status.EXL、Cause.ExcCode、Cause.BD、EPC字段

如果Status.EXL = 1（发生中断、异常时在处理异常中）则：

Interrupt0~ Interrupt7、Sys、Ri、Ov、Tr仅修改Cause.ExcCode即可。

否则：

Interrupt0~ Interrupt7异常：

Status.EXL = 1

Cause.ExcCode = 0（编码见CP0设计部分）

位于延迟槽中（is\_in\_delayslot == 1’b1）

Cause.BD = 1

EPC = current\_inst\_addr – 4

不位于延迟槽中

Cause.BD = 0

EPC = current\_inst\_addr

Sys：

Status.EXL = 1

Cause.ExcCode = 8（编码见CP0设计部分）

位于延迟槽中（is\_in\_delayslot == 1’b1）

Cause.BD = 1

EPC = current\_inst\_addr – 4

不位于延迟槽中

Cause.BD = 0

EPC = current\_inst\_addr

Ri：

Status.EXL = 1

Cause.ExcCode = 10（编码见CP0设计部分）

位于延迟槽中（is\_in\_delayslot == 1’b1）

Cause.BD = 1

EPC = current\_inst\_addr – 4

不位于延迟槽中

Cause.BD = 0

EPC = current\_inst\_addr

Ov：

Status.EXL = 1

Cause.ExcCode = 12（编码见CP0设计部分）

位于延迟槽中（is\_in\_delayslot == 1’b1）

Cause.BD = 1

EPC = current\_inst\_addr – 4

不位于延迟槽中

Cause.BD = 0

EPC = current\_inst\_addr

Tr：

Status.EXL = 1

Cause.ExcCode = 13（编码见CP0设计部分）

位于延迟槽中（is\_in\_delayslot == 1’b1）

Cause.BD = 1

EPC = current\_inst\_addr – 4

不位于延迟槽中

Cause.BD = 0

EPC = current\_inst\_addr

eret：

Status.EXL = 0

1. **涉及部件说明**

1、except\_detect1部件

作用说明：译码阶段异常检测，except\_detect1接收来自decoder的异常信号，判断是否存在syscall异常，异常返回以及无效指令，并在excepttype记录异常信息后向后传递，进行异常信息收集。

|  |  |  |
| --- | --- | --- |
| syscall(1) | except\_detect1 |  |
|  |  |
| eret(1) | (32)excepttype |
|  |  |
| invalid\_inst(1) |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度(bit) | 输入/输出 | 作 用 |
| 1 | syscall | 1 | 输入 | 由译码器传入，指示是否是syscall指令 |
| 2 | eret | 1 | 输入 | 由译码器传入，指示是否是eret指令 |
| 3 | invalid\_inst | 1 | 输入 | 由译码器传入，指示是否是无效指令 |
| 4 | excepttype | 32 | 输出 | 异常信息记录数据 |

2、except\_detect2部件

作用说明：执行阶段异常检测，except\_detect2接收来自alu的flag信息，判断是否产生自陷异常以及是否产生溢出异常，并在excepttype记录异常信息后向后传递，继续进行异常信息收集。

|  |  |  |
| --- | --- | --- |
| lf(1) | except\_detect2 |  |
| of(1) |  |
| zf(1) |  |
| condition(3)  trap(1)  overflow\_detect(1) | (32)excepttype\_out |
| Excepttype\_in(32) |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度(bit) | 输入/输出 | 作 用 |
| 1 | lf | 1 | 输入 | Alu传出的lf信号，表示less flag |
| 2 | of | 1 | 输入 | Alu传出的of信号，表示overflow |
| 3 | zf | 1 | 输入 | Alu传出的zf信号，表示zero flag |
| 4 | condition | 32 | 输入 | Trap指令条件指示，使用与branch指令相同的数据流与编码：  001：==  011：>=  110：<  010：!= |
| 5 | trap | 1 | 输入 | 说明是否是自陷指令 |
| 6 | overflow\_detect | 1 | 输入 | 说明是否需要检测溢出 |
| 7 | excepttpe\_in | 32 | 输入 | 由except\_detect1传出的异常数据（实际连接是从ID/EX传入的），继续进行异常记录 |
| 8 | excepptype\_out | 32 | 输出 | 所有异常的记录信息 |

3、interrupt\_except\_handle部件

作用说明：访存阶段中断检测以及异常中断的优先级判断，输出此次处理的异常中断类型编码，interrupt\_except\_handle接收来自cp0中status、cause寄存器的值，以及上一阶段继续收集的异常信息，判断优先级后输出要处理的异常编码，同时撤销对于内存的访问请求（出现异常的代码不允许对内存进行访问）。

|  |  |  |
| --- | --- | --- |
| cp0\_status\_i(32) | interrupt\_except\_handle | (32)excepttype\_o |
|  |  |
| cp0\_cause\_i(32) | (1)store\_enable |
| excepttype\_i(32) | (1)load\_enable |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度(bit) | 输入/输出 | 作 用 |
| 1 | cp0\_status\_i | 32 | 输入 | 由cp0传入的status寄存器值，用于中断产生判断 |
| 2 | cp0\_cause\_i | 32 | 输入 | 由cp0传入的cause寄存器值，用于中断产生判断 |
| 3 | excepttype\_i | 32 | 输入 | 由except\_detect2传出的异常数据（实际连接是从EX/MEM传入的），继续进行中断记录后，用于判断中断异常优先级 |
| 4 | excepttype\_o | 32 | 输出 | 此次处理的异常或者中断类型，输入cp0以及Control Unit进行异常处理 |
| 5 | store\_enable | 1 | 输出 | store类型指令访存的使能信号，若产生异常或者中断，不允许将数据写入内存 |
| 6 | load\_enable | 1 | 输出 | Load类型指令访存的使能信号，若产生异常或者中断，不允许从内存读数据 |

1. **异常中断检测整体数据通路说明（红色部分）**

E:\学习文档\大三上\计算机组成与设计\流水线\设计文档\cu_cp0_异常交互.emf

部件对应说明：except\_detect1对应图中“异常检测1”，except\_detect2对应图中“异常检测2”，interrupt\_except\_handle对应图中“中断检测”。

异常检测数据流说明：(红色部分对应异常中断检测，蓝色部分对应mfc0/mtc0数据流)

异常检测由decoder译码开始，将译码信息（syscall/eret/无效指令）输入异常检测1，异常检测1记录这三种异常信息，将异常以及传到ID/EX流水线段寄存器；之后继续将该信息传到异常检测2，异常检测2接收ALU的信息，并判断是否检测溢出以及自陷异常，如果需要检测，则将出现异常的信息继续记录到excepttype中，向下传递到EX/MEM流水线段寄存器；中断检测读取该信息，并读取cp0记录的此时的中断产生信息，同时根据cp0中的中断使能以及中断屏蔽信息，判断产生哪种中断，并继续记录在excepttype中。继续在中断检测中进行优先级判断之后，得到此阶段真正需要处理的中断或者异常（产生多个中断异常但每次只处理一个），输出该中断或者异常的编码信息到cp0以及Control Unit中，cp0进行此次指令的pc记录（用于eret或者异常处理程序处理）以及异常类型记录（用于异常处理程序），Control Unit进行中断异常处理，即将之前的指令冲刷掉，输出cu\_vector为异常处理程序起始地址，并选择pc为该地址，使pc开始执行异常处理程序。

异常处理程序保存目前的寄存器状态，再根据cp0中记录的异常中断类型来进一步选择相应的处理类型，同时使用保存的寄存器值作为判断信息。处理完毕之后，异常处理程序恢复保存的寄存器值，再通过cp0记录的此次异常指令的pc记录，来跳转到原本发生异常的指令，继续开始执行原来的指令。