64位多核 MIPS 异常和中断结构

1. 概述

MIPS 统称异常（同步事件）和中断（异步事件）为例外 (Exception)

引入异常则是为了解决处理器运行过程中的一些意外情形，比如执行流中有非法指令（无法被处理器辨识的指令），访问了没有映射的虚拟地址等等。

中断的引入则是提供一种 IO 设备请求处理器服务的一种通讯机制；往往是由外部设备主动发起（给处理器的中断引脚一个信号），处理器收到信号后跳转到特定入口处执行。

处理器运行在用户态的话，例外发生时，则会进行模式切换（即进入特权级，可以执行管理系统资源的特权指令，比如 mfc0/mtc0），因此中断和异常也可看作是一些事件（主动及意外）请求特权服务。因为现代操作系统都运行在特权级下，拥有并管理系统资源，因此中断和异常实际是请求操作系统服务。

因为例外发生时总会伴随模式切换，即进入特权模式，因此对于程序需要主动进入特权模式的情形（比如用户态程序通过系统调用请求操作系统服务），MIPS 引入一条 'syscall' 指令来触发系统调用例外，一旦该指令被执行，处理器即切换到特权级并进入“系统调用例外”

处理器的一个例外过程，多数情形下（非嵌套）其实就相当于一个伴随模式切换的过程调用。例外发生后，MIPS 在置 CP0 寄存器 Status[EXL] = 1，并将当前 PC 值存入 EPC (Exception PC，即指向发生例外的指令)，就从当前指令流跳转到另外一个指令流，MIPS 硬件不负责保存上下文，因此软件先保存上下文，服务完成后恢复上下文，再执行指令 'eret' 例外返回（eret 所作的操作为：将 EPC/ErrorEPC 的值置入 PC，同时清除 CP0_Status 寄存器的 EXL 位）。MIPS 下，只要 Status 寄存器的 EXL (exception) 位为 1 即表示处理器进入例外，处于特权模式下。

首先看一个系统调用异常发生时的现场过程：

用户执行 'syscall' 指令，触发系统调用异常，MIPS 这样响应：

a. 将引起异常的 syscall 指令所在地址压入 EPC

b. 置 STATUS 的 EXL 位为 1（其他位不变）

c. 设置 CAUSE 的 ExcCode 位为 8，指明这是其他例外里的系统调用异常

d. 跳转到其它例外入口

CPU 的 STATUS[EXL] 被置为 1，这个意味着：

I. CPU 自动进入内核模式（忽略 STATUS[KSU] 位），禁止中断（忽略 STATUS[IE] 位）

II. TLB Refill 例外入口使用其他例外入口而非原 TLB Refill 入口

III. 当又出现新例外时，CPU 将不会重新设置 EPC 和 CAUSE[BD]

一个中断过程： 特权级迁移 例外向量入口 异常类型、优先级 中断特殊性

由于 MIPS 版本众多，光官方的规范版本就有 MIPS32 R1, MIPS32 R2, MIPS64 R1, MIPS64 R2 四个版本，全覆盖地讨论实在没有必要，下文单核将以 Loongson 2E 为范例（类 MIPS64 R1），多核将以目前使用较多、并且 Kernel Mainline 已有支持的 Cavium Octeon CN38XX 系列为例 (MIPS64 R2) 讨论。

2. 高优先级例外类型和入口

所谓高优先级例外，就是需要 MIPS 尽可能快地响应并执行例外处理函数的例外。

MIPS 高优先级例外有 4 类，分别为：

Vec0: 冷启动、热重启、非屏蔽中断 (Reset)

Vec1: TLB Refill

Vec2: Cache Error

Vec3: 其它例外

他们在 MIPS 上，有固定的入口地址，一旦例外发生，即跳转到这个地址，迅速处理。

Reset 类例外容易理解，一旦这类发生，将伴随系统启动，需要即刻响应。固定入口地址处，实际上就是 bootloader 的第一条指令，也是存放 bootloader 的 flash 之所在。

由于 MIPS 采用的是软件 TLB，即需要 OS 读取内存中的页表来填充 TLB。这个对系统的性能影响很大，因此也需要第一时间处理。

Cache 发生错误时，往往内部数据就不再可信，因此需要第一时间处理。

其他例外包括中断，系统调用，地址错误，非法指令等等例外都是

2.1 正常运行模式下例外入口

2.1.1 MIPS64 R1 情形

MIPS 设计时固定了四大类例外的入口，正常运行情形下（非初始化情形，BEV=0）：

TLB Refill 时，PC 的值为 0xFFFF FFFF 8000 0000，对应物理地址 0x00 处，访问的数据会进入 Cache (32 bit address space)

xTLB Refill 时，PC 的值为 0xFFFF FFFF 8000 0080，对应物理地址 0x80 处，访问的数据会进入 Cache (64 bit address space)

Cache Error 时，PC 的值为 0xFFFF FFFF a000 0100，对应物理地址 0x100 处，访问的数据不会进入 Cache

其它例外时，PC 的值为 0xFFFF FFFF 8000 0180，对应物理地址 0x180 处，访问的数据会进入 Cache

MIPS64 通常是用 xTLB Refill 例外入口，为 0xFFFF FFFF 8000 0080，其处理程序的体积最大为 0x80 字节（MIPS32 为 0x8000 0000，容量为 0x80 字节）,loongson 2E 比较特殊，他的入口只用了 32 位地址空间的，始终为 0xFFFF FFFF 8000 0000

Cache Error 例外，因为 Cache 已不能相信，因此只能使用不走 Cache 的虚拟地址 0xFFFF FFFF a000 0100 置入 PC 中。其处理程序的体积最大为 0x80 字节

其它例外，入口在 0xFFFF FFFF 8000 0180，MIPS 的其他例外都进入此入口，由软件根据寄存器中的通用例外类型，进入到对应的处理函数。其处理程序的体积一般不超过 0x80 字节

2.1.2Cavium Octeon (MIPS64 R2) 情形

对于像 Cavium Octeon 这类多核处理器，他有 16 个 CPU 核，在一些特殊应用环境下（比如其中 1 个核跑一个 Linux 作为控制面，其他 15 个核，每个核只跑一个应用作为数据面），不同的核就需要不同的例外入口基地址。为了这个原因，MIPS 在 MIPS64 R2 里引入了 CP0_EBase 寄存器，用于存放例外入口的基地址。

MIPS64 R2 规定，CP0_EBase 只在 CP0_STATUS[BEV]=0 时，即在正常运行时有效，BEV=1 的上电启动状态，例外的基地址还是原来的地址。

CP0_EBase 的默认值为 0x8000 0000，32bit 的值在 64bit 下，高位自动扩展：0xFFFF FFFF 8000 0000，这样就能和 MIPS64 R1 兼容，即使用户无视这个寄存器，原系统也能在 R2 上正常运行。

有了 CP0_EBase 后，不同的核就可以有自己的例外入口，这样就可以各自跑自己的系统，甚至只跑一个应用。

因此在 MIPS64 R2 下，BEV ＝ 0 时只规定四个固定例外入口的偏移：

TLB Refill 时，偏移为 0x00 (32 bit address space)

xTLB Refill 时，偏移为 0x80 (64 bit address space)

Cache Error 时，偏移为 0x100

其它例外时，偏移为 0x180实际的入口为 0xFFFF FFFF 0000 0000 + (cp0_ebase & 0x3FFF F000)

Cavium Octeon 上 Bootloader 将异常的基地址设为 0xFFFF FFFF 8000 1000

2.2上电初始化时例外入口

下面这一段与 Linux Kernel 关系不大，但与 Bootloader 关系密切，请酌情选读：

对于 MIPS 刚上电的初始化阶段（CP0_STATUS[BEV]=1），因为 Cache 状态未知，不能使用，所以所有的例外入口虚地址都得使用数据不进入 Cache 的地址段。因此 MIPS 规定，BEV=1 时，例外向量入口为：

冷启动、热重启、非屏蔽中断，入口为 0xFFFF FFFF BFC0 0000

TLB Refill，入口为 0xFFFF FFFF BFC0 0200

Cache Error，入口为 0xFFFF FFFF BFC0 0300

其它例外，入口为 0xFFFF FFFF BFC0 0380

特别地，冷启动、热重启、非屏蔽中断例外时，PC 的值都为 0xFFFF FFFF BFC0 0000（MIPS32 为 0xBFC0 0000），即当发生这类例外时，处理器总是跳转到这个地址继续执行，其固定映射到的物理地址为 0x1FC0 0000 处。这个地址常常关联到放置 Bootloader 的内部 flash，每次一 reset，就会触发这个例外，跳转到 0x1FC0 0000，去取 Bootloader 的第一条指令。

可以看到在初始化阶段 (CP0_STATUS[BEV]＝1，bootloader 最初执行时)，4 类例外处理程序的最大允许长度分别为：0x200, 0x100, 0x80, 通常不超过 0x80 字节

注意 BEV=1 的时间很短，一旦 bootloader 完成初始化，其就将 BEV 置为 0，进入系统正常运行模式

3. 其它例外的子类型

当发生的例外不是 Reset, TLB Refill, Cache Error 时，MIPS 进入其它例外入口，对这个其它例外的区分，即 MIPS 怎么告诉 OS， 系统调用是进系统调用的处理函数，地址错误是进另外的处理函数。MIPS 如何做的呢？

他是在 CP0 的 STATUS 寄存器里，保留了 5 位 (STATUS[6:2]) 作为 ExcCode，即例外 Code。不同的其它例外有不用的 ExcCode:

0 Int 中断

1 Mod TLB 修改异常

2 TLBL TLB 读异常

3 TLBS TLB 写异常

4 AdEL 读地址错误异常

5 AdES 写地址错误异常

.....

8 Sys 系统调用异常

.....

31 保留

这样 OS 就可以用 ExcCode 作为一个索引值，去索引一个函数数组，啥样的例外进啥样的处理函数。至于他们怎么传递参数和返回值，后面详细讨论。

MIPS Linux 下，中断和系统调用的处理函数为 handle_int 和 handle_sys

4. 中断的特殊结构

MIPS CPU 收到硬件中断请求后，会跳到其他例外入口，此时 CP0_STATUS 的 ExcCode 的值为 0，此只能描述“有了中断”，尚不能确定中断来自哪个中断控制器亦或就是 CPU 内部的硬件中断，比如 Timer 中断或者 Performace Counter 溢出中断

因此 MIPS 把 CP0_CAUSE 的 15:8 位留给了 8 个中断，可以描述 8 个中断的到来 (Pending)，对应位为 1，则表示相应中断到来：

Bit | Name | Meaning
15 | IP7 | Hardware interrupt 5
14 | IP6 | Hardware interrupt 4
13 | IP5 | Hardware interrupt 3
12 | IP4 | Hardware interrupt 2
11 | IP3 | Hardware interrupt 1
10 | IP2 | Hardware interrupt 0

09 | IP1 | Request software interrupt 1
08 | IP0 | Request software interrupt 0

相对应的，CP0_STATUS[15:8] 为 IM7～IM0，是为 IP7～IP0 的中断 Mask 位

IP1～IP0 对应 CP0_CAUSE[9:8]，留给软中断

IP2～IP7 则给硬件中断

在 MIPS64 R1 里，IP7 留给 Timer 或者 Performace Counter，至于是哪个，实现时自己取舍。Loongson 2E 实现时，把 IP7 给了 Timer；Performace Counter 用 IP6。

而在 MIPS64 R2 里，IP7 可以同时给 Timer 和 Performace Counter。当 IP7 被置位时，OS 可以检查 CP0_CAUSE[30] 和 CP0_CAUSE[26]，前者为 1，表示 TI (Timer Interrupt)；位 26 为 1，则表示是 PCI (Performace Counter Interrupt)

通常来说， IP7～IP2 会对应 6 个电气信号引脚。在 2E 上对应 Int#[5..0]，当然 Int#[1:0] 保留未用

各外围设备都可以直接把中断请求线接到 IP2～IP6 上，但现代系统上外围设备五花八门，种类繁多，几根线根本不够用，因此一般都是外围设备的中断先汇总到中断控制器，中断控制器再单独向 CPU 请求中断。

比如 Loongson 2E 上，一个类 8259A 的中断控制器（负责键盘、声卡、IDE 等设备的中断），链接到 IP5

北桥内的中断控制器（负责 PCI 上的中断，如网卡、USB 等）则链接到 IP2

Cavium Octeon 上 CIU0（负责 UART0/1, MailBox0/1, USB0, MII0, WorkQueue 等）链接到 IP2；CIU1（负责 UART2, USB1, MII1, NAND 等）连接到 IP3。16 个核，CIU0 都连接到每个核的 IP2，CIU1 也都连接到每个核的 IP3

不管 IP2, IP3 直接挂设备还是挂中断控制器，OS 都可以区分到底是哪个具体设备的中断，对中断控制器，只要访问一下中断控制器的内部状态寄存器，就能确定是哪个具体设备的中断

也可以多个设备/中断控制器接到同一个 IP2，如果同时发生，这时候 OS 处理时就要负责判断优先级了，哪个先执行哪个后执行。当然有人要说了，这样不就把中断控制器省略了吗？理论上是可以的，但当设备繁多到上百后，原来控制器硬件做的事都由软件来做对性能是一个不小的影响，而且可扩展性也是一个问题。微型系统上用用可以，大一点的系统就不合适了。

在 2E 上，一个用户敲击键盘的行为，中断流程是这样的：

1. 用户击键后，键盘控制器8042产生中断，这个中断链接到 (route) I8259A，并由 8259A 在 CPU 的中断引脚上发起中断请求

2. CPU 自动设置 CAUSE 的 ExcCode 位为0，IP5 为1，并跳转到其他例外入口 0xFFFFFFFF 80000180

3. 位于其他例外入口处的简单异常处理程序，根据 ExcCode 的值索引例外处理表 (exception_handlers)，获取到0号例外的处理程序是 handle_int，并跳转过去

4. handle_int 根据 CAUSE 之 IP 位的值跳转到中断控制器 8259A 相关的中断处理函数 i8259_irqdispatch()

5. i8259_irqdispatch() 读取 8259A 之 In-Service Register (ISR, 注意与 x86的差异，x86是由 8259A 主动将中断号送上数据总线的)，通过简单的计算得到中断号，进而调用 do_IRQ 进入相应的键盘中断处理函数

类似 Cavium Octeon 的多核处理器，往往都在 CPU 内部实现一个中央中断控制器 (CIU), 来负责所有中断的路由，哪一个设备中断路由到哪一个核，都是由中央中断控制器控制，并且是可配置的，一般 OS 也是能更改的，即 OS 可以根据当前核的中断负载情形，对中断路由进行动态调整。Cavium Octeon 的详细情形，后文讨论。

5. 摘要

四个高优先级例外，对应四个固定入口。前三个 Reset，TLB Refill，Cache Error 需要系统尽可能快地进入处理函数，因此他们的处理函数直接位于固定入口处。

所有其他例外共享第四个入口，统称一般例外 (General Exception)。他的处理函数区分具体哪个例外是通过 CP0_STATUS 的 ExcCode

中断例外的 ExcCode 为 0，他的处理函数区分具体的中断是通过 CP0_STATUS 的 IP7 ~ IP0，外围的中断控制器固定链接到其中的一位 IPn (n = 6 ~ 2)，中断控制器收到中断请求时，他会向上传递这个请求，IPn 会被置 1。这样就能进入到具体的中断控制器的处理函数中，他再访问一下中断控制器内部的 ISR 就能获取到是哪个具体的设备，从而进一步进入设备中断处理函数。