[詳解RISC v中斷](https://www.cnblogs.com/harrypotterjackson/p/17548837.html)

**閱讀目錄**

* [常規中斷](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label0)
* [U 與 S 之間的切換](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label1)
* [S 與 M 之間的切換](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label2)
* [地址空間佈局](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label3)
* [跳板機制](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label4)
* [第一次的 stvec 是如何設定的](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label5)
* [PLIC](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label6)
* [ACLINT](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label7)
* [CLIC](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label8)
* [時鐘中斷相關的暫存器](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label9)
* [時鐘中斷的基本處理過程](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label10)
* [xv6 的實現](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label11)
* [Linux 的時鐘中斷的實現](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label12)
* [QEMU 的時鐘中斷的邏輯](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label13)
* [參考文獻](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label14)
* [中斷髮送](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label15)
* [中斷處理](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label16)
* [參考資料](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_label17)

**聲明**

本文為本人原創，未經許可嚴禁轉載。部分圖源自網路，如有侵權，聯絡刪除。

**RISC-V 中斷與異常**

trap（陷阱）可以分為異常與中斷。在 RISC v 下，中斷有三種來源：software interrupt、timer interrupt（顧名思義，時鐘中斷）、external interrupt。

有同學可能見過 NMI，但是這是一種中斷類型而非中斷來源。Non-maskable interrupt，不可遮蔽中斷，與之相對的就是可遮蔽中斷。NMI 都是硬體中斷，只有在發生嚴重錯誤時才會觸發這種類型的中斷。

有同學可能接觸過 Linux 中的軟中斷，即 softirq，但是請注意 software interrupt 與 softirq 是完完全全不一樣的。如果你沒有接觸過 softirq 就請現在就暫停本文去瞭解一下，否則把 Linux 中的 softirq 與 software interrupt 搞混是會貽笑大方的。

本文將全面介紹 RISC v 下的中斷髮送與處理、軟體中斷、使用者態中斷和特權級轉換，並結合 xv6 核心、rcore、Linux 核心等實現進行介紹。

**與中斷有關的暫存器**

下面所述的都是軟體中斷、外部中斷和異常相關的內容，時鐘中斷比較特殊將單獨介紹。

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**常規中斷[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "3695266693)**

M-mode 的暫存器

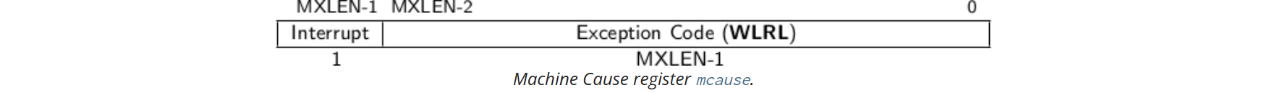
mstatus，mtvec，medeleg，mideleg，mip，mie，mepc，mcause，mtval

S-mode 的暫存器

sstatus，stvec，sip，sie，sepc，scause，stval，satp

在後文中，我們可能會有 xstatus , xtvec 等的寫法，其中 x 表示特權級 m 或者 s 或者 u（u 僅僅在實現了使用者態中斷的 CPU 上存在）。

**mcause[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "2964591593)**

[](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210012313-359133103.png)

如果陷阱是由中斷引起的，則 mcause 暫存器中的“Interrupt”位被設定。Exception Code 欄位用於標識最後一個異常或中斷的程式碼。下表列出了可能的機器級異常程式碼。異常程式碼是 WLRL 欄位，因此僅保證包含受支援的異常程式碼。

(PS: 讀者可能疑惑為啥在 mcause 中會存在 Supervissor software interrupt [TODO])

**mstatus[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "1791177573)**

[一張含有 日曆 的圖片

自動產生的描述](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210012932-1184025042.png)

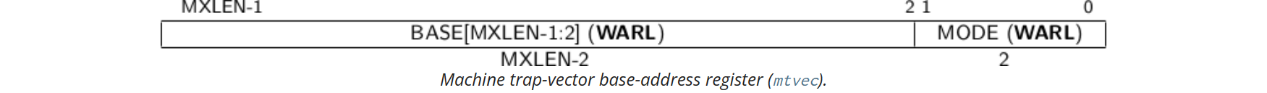
MIE 與 SIE 是全域中斷使能位。當 xIE 為 1 時，允許在 x 特權級發生中斷，否則不允許中斷。

當 hart 處於 x 特權級時，當 xIE 為 0 時，x 特權級的中斷被全部停用，否則被全部啟用。當 xIE 為 0 時，對於任意的 w<x，w 特權級的中斷都是處於全域停用狀態。對於任意的 y>x，y 特權級的中斷默認處於全域啟用狀態，無論 xIE 是否為 1。

為支援巢狀陷阱，每個可以響應中斷的特權模式 x 都有一個兩級中斷使能位和特權模式堆疊。xPIE 保存陷阱之前活動的中斷使能位的值，xPP 保存之前的特權模式。xPP 欄位只能保存 x 及以下特權模式，因此 MPP 為兩位寬，SPP 為一位寬。當從特權模式 y 進入特權模式 x 時，xPIE 設定為 xIE 的值；xIE 設定為 0；xPP 設定為 y。對於 MPP，可以設定的值有 0b00（使用者模式），0b01（S-mode），0b10(reserved)，0b11(M-mode)

在 M 模式或 S 模式中，使用 MRET 或 SRET 指令返回陷阱。執行 xRET 指令時，將 xIE 設定為 xPIE；將 xPIE 設定為 1；假設 xPP 值為 y，則將特權模式更改為 y；將 xPP 設定為 U（如果不支援使用者模式，則為 M）。如果 xPP≠M，則 xRET 還會設定 MPRV=0。

**mtvec[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "916841757)**

[](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210013404-19738400.png)

mtvec 記錄的是異常處理函數的起始地址。BASE 欄位中的值必須始終對齊於 4 位元組邊界，並且 MODE 設定可能會對 BASE 欄位中的值施加額外的對齊約束。

MODE 目前可以取兩種值：

[一張含有 文字 的圖片

自動產生的描述](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210013684-694188924.png)

如果 MODE 為 0，那麼所有的異常處理都有同一個入口地址，否則的話異常處理的入口地址是 BASE+4\*CAUSE。（cause 記錄在 xcause 中）

要求異常處理函數的入口地址必須是 4 位元組對齊的。

**medeleg 與 mideleg[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "380023569)**

默認情況下，各個特權級的陷阱都是被捕捉到了 M-mode，可以通過程式碼實現將 trap 轉發到其它特權級進行處理，為了提高轉發的性能在 CPU 等級做了改進並提供了 medeleg 和 mideleg 兩個暫存器。

medeleg （machine exception delegation）用於指示轉發哪些異常到 S-mode；mideleg(machine interrupt delegation)用於指示轉發哪些中斷到 S-mode。

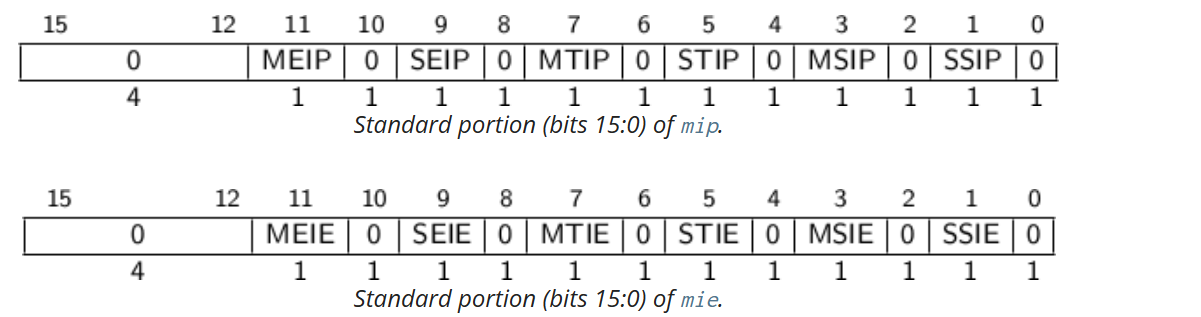
當將陷阱委託給 S 模式時，scause 暫存器會寫入陷阱原因；sepc 暫存器會寫入引發陷阱的指令的虛擬地址；stval 暫存器會寫入特定於異常的資料；mstatus 的 SPP 欄位會寫入發生陷阱時的活動特權級；mstatus 的 SPIE 欄位會寫入發生陷阱時的 SIE 欄位的值；mstatus 的 SIE 欄位會被清除。mcause、mepc 和 mtval 暫存器以及 mstatus 的 MPP 和 MPIE 欄位不會被寫入。

假如被委託的中斷會導致該中斷在委託者所在的特權級遮蔽掉。比如說 M-mode 將一些中斷委託給了 S-mode，那麼 M-mode 就無法捕捉到這些中斷了。

**mip 與 mie[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "2328490179)**

mip 與 mie 是分別用於保存 pending interrupt 和 pending interrupt enable bits。每個中斷都有中斷號 i（定義在 mcause 表中），每個中斷號如果被 pending 了，那麼對應的第 i 位就會被置為 1. 因為 RISC v spec 定義了 16 個標準的中斷，因此低 16bit 是用於標準用途，其它位則\*台自訂。

如下圖所示是低 16bit 的 mip 與 mie 暫存器。其實比較好記憶，只需要知道 mcause 中的中斷源即可。例如 SSIP 就是 supervisor software interrupt pending, SSIE 就是 supervisor software interrupt enable。

[](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210014053-1431446270.png)

如果全域中斷被啟用了，且 mie 和 mip 的第 i 位都為 1，那麼中斷 i 將會被處理。默認情況下，如果當前特權級小於 M 或者當前特權級為 M 切 MIE 是 1 的話，全域中斷就是被啟用的；如果 mideleg 的第 i 位為 1，那麼噹噹前特權級為被委託的特權級 x（或者是小於 x），且 mstatus 中的 xIE 為 1 那麼就認為是全域中斷是被啟用的。

暫存器 mip 中的每個位都可以是可寫的或唯讀的。當 mip 中的第 i 位可寫時，可以通過向該位寫入 0 來清除掛起的中斷 i。如果中斷 i 可以變為掛起但 mip 中的位 i 是唯讀的，則實現必須提供一些其他機制來清除掛起的中斷。如果相應的中斷可以變為掛起，則 mie 中的位必須是可寫的。不可寫的 mie 位必須硬連線為零。

位 mip .MEIP 和 mie .MEIE 是 M-mode 外部中斷的中斷掛起和中斷允許位。 MEIP 在 mip 中是唯讀的，由\*台特定的中斷控製器設定和清除。

位 mip .MTIP 和 mie .MTIE 是 M-mode 定時器中斷的中斷掛起和中斷允許位。 MTIP 在 mip 中是唯讀的，通過寫入對應到記憶體的 mtimecmp 來清除。

位 mip .MSIP 和 mie .MSIE 是機器級軟體中斷的中斷掛起和中斷允許位。 MSIP 在 mip 中是唯讀的，通過訪問記憶體對應控制暫存器寫入，遠端 harts 使用這些暫存器來提供 M-mode 處理器間中斷。 hart 可以使用相同的記憶體對應控制暫存器寫入自己的 MSIP 位。

如果實現了 S-mode，位 mip .SEIP 和 mie .SEIE 是 S-mode 外部中斷的中斷掛起和中斷允許位。 SEIP 在 mip 中是可寫的，並且可以由 M 模式軟體寫入以向 S 模式指示外部中斷正在掛起。此外，\*台級中斷控製器（PLIC）可以生成 S-mode 外部中斷。SEIP 位是可寫的，因此需要根據 SEIP 和外部中斷控製器的訊號進行邏輯或運算的結果，來判斷是否有掛起的 S-mode 外部中斷。當使用 CSR 指令讀取 mip 時， rd 目標暫存器中返回的 SEIP 位的值是 mip.SEIP 與來自中斷控製器的中斷訊號的邏輯或。但是，CSRRS 或 CSRRC 指令的讀取-修改-寫入序列中使用的值僅包含軟體可寫 SEIP 位，忽略來自外部中斷控製器的中斷值。

*SEIP 欄位行為旨在允許更高權限層乾淨地模擬外部中斷，而不會丟失任何真實的外部中斷。因此，CSR 指令的行為與常規 CSR 訪問略有不同。*

如果實現了 S-mode， mip .STIP 和 mie .STIE 是 S-mode 定時器中斷的中斷掛起和中斷允許位。 STIP 在 mip 中是可寫的，並且可以由 M 模式軟體編寫以將定時器中斷傳遞給 S 模式。

位 mip .SSIP 和 mie .SSIE 是管理級軟體中斷的中斷掛起和中斷允許位。 SSIP 在 mip 中是可寫的。

S-mode 的 interprocessor interrrupts 與實現機制有關，有的是通過呼叫 System-Level Exception Environment(SEE)來實現的，呼叫 SEE 最終會導致在 M-mode 將 MSIP 位置為 1. 我們只允許 hart 修改它自己的 SSIP bit，不允許修改其它 hart 的 SSIP，這是因為其它的 hart 可能處於虛擬化的狀態、也可能被更高的 descheduled。因此我們必須通過呼叫 SEE 來實現 interprocessor interrrupt。M-mode 是不允許被虛擬化的，而且已經是最高特權級了，因此可以直接修改其它位的 MSIP，通常是使用非緩衝 IO 寫入 memory-mapped control registers 來實現的，具體依賴於\*台的實現機制。

多個同時中斷按以下優先順序遞減順序處理：MEI、MSI、MTI、SEI、SSI、STI。異常的優先順序低於所有中斷。

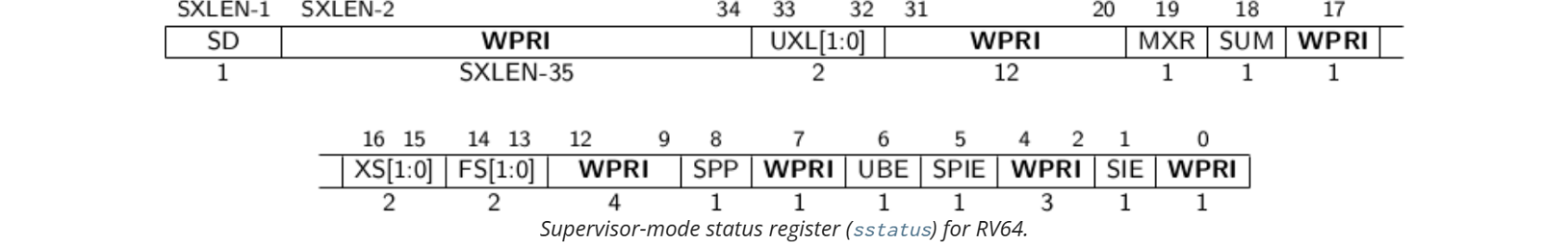
**mepc[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "2620262275)**

當 trap 陷入到 M-mode 時，mepc 會被 CPU 自動寫入引發 trap 的指令的虛擬地址或者是被中斷的指令的虛擬地址。

**mtval[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "1705224279)**

當 trap 陷入到 M-mode 時，mtval 會被置零或者被寫入與異常相關的資訊來輔助處理 trap。當觸發硬體斷點、地址未對齊、access fault、page fault 時，mtval 記錄的是引發這些問題的虛擬地址。

**stastus[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "2325222487)**

[](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210014505-699545048.png)

與中斷相關的欄位是 SIE、SPIE、SPP。

SPP 位指示處理器進入 supervisor 模式之前的特權等級。當發生陷阱時，如果該陷阱來自使用者模式，則 SPP 設定為 0；否則設定為 1。當執行 SRET 指令從陷阱處理程序返回時，如果 SPP 位為 0，則特權等級設定為使用者模式；如果 SPP 位為 1，則特權等級設定為 supervisor 模式；然後將 SPP 設定為 0。

SIE 位在 supervisor 模式下啟用或停用所有中斷。當 SIE 為零時，在 supervisor 模式下不會進行中斷處理。當處理器在使用者模式下執行階段，忽略 SIE 的值，並啟用 supervisor 等級的中斷。可以使用 sie 暫存器 來停用單個中斷源。

SPIE 位指示陷入 supervisor 模式之前是否啟用了 supervisor 等級的中斷。當執行跳轉到 supervisor 模式的陷阱時，將 SPIE 設定為 SIE，並將 SIE 設定為 0。當執行 SRET 指令時，將 SIE 設定為 SPIE，然後將 SPIE 設定為 1。

**其它 s 特權級暫存器[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "865036678)**

stvec, sip, sie,sepc, scause, stval 與 m-mode 的相應暫存器區別不大，讀者可自行參閱 RISC v 的 spec。

satp 比較特殊，在 M-mode 沒有對應的暫存器，因為 M-mode 沒有分頁，satp 記錄的是根頁表實體位址的頁幀號。在從 U 切換到 S 時，需要切換頁表，也即是切換 satp 的根頁表實體位址的頁幀號。

**特權級轉換**

我在這裡只介紹了 U 和 S 之間的切換，其實 S 和 M 之間的切換過程也是一樣的，只不過使用到的暫存器不一樣了而已。比如說保存 pc 的寄存，S 保存 U 的 pc 值使用的是 sepc，M 保存 S 的 pc 使用的是 mepc。此外，U 切換到 S 時一般需要切換頁表，而從 S 切換到 M 時不需要切換頁表，因為 M 沒有實現分頁，也沒有 matp 暫存器（頁表根地址儲存在 satp 暫存器中，所以我這裡胡謅了個 matp）。

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**U 與 S 之間的切換[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "4014758991)**

**U 切換到 S**[**#**](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#2881633038)

當執行一個 trap 時，除了 timer interrupt，所有的過程都是相同的，硬體會自動完成下述過程：

1. 如果該 trap 是一個裝置中斷並且 sstatus 的 SIE bit 為 0，那麼不再執行下述過程
2. 通過置零 SIE 停用中斷
3. 將 pc 複製到 sepc
4. 保存當前的特權級到 sstatus 的 SPP 欄位
5. 將 scause 設定成 trap 的原因
6. 設定當前特權級為 supervisor
7. 複製 stvec（中斷服務程序的首地址）到 pc
8. 開始執行中斷服務程序

CPU 不會自動切換到核心的頁表，也不會切換到核心棧，也不會保存除了 pc 之外的暫存器的值，核心需要自行完成。對於Linux而言，核心空間與使用者態空間是使用的同一套頁表，不需要切換頁表。詳情可以參考使用者態處理程序的虛擬記憶體佈局。核心空間一般位於處理程序的高虛擬地址空間。

對於沒有開啟分頁，如何切換特權級可以參考：[實現特權級的切換 - rCore-Tutorial-Book-v3 3.6.0-alpha.1 文件](https://rcore-os.cn/rCore-Tutorial-Book-v3/chapter2/4trap-handling.html)

如果啟用了分頁，當陷入到 S 模式時，CPU 沒有切換頁表（換出處理程序的頁表，換入核心頁表），核心需要自行切換頁表，參考：[核心與應用的地址空間 - rCore-Tutorial-Book-v3 3.6.0-alpha.1 文件](https://rcore-os.cn/rCore-Tutorial-Book-v3/chapter4/5kernel-app-spaces.html" \t "_blank) 和 [基於地址空間的分時多工 - rCore-Tutorial-Book-v3 3.6.0-alpha.1 文件](https://rcore-os.cn/rCore-Tutorial-Book-v3/chapter4/6multitasking-based-on-as.html) 。

其實切換頁表的過程也很簡單，只需要將核心的頁表地址寫入 satp 暫存器即可。

在執行中斷服務例程時還需要首先判斷 sstatus 的 SPP 欄位是不是 0，如果是 0 表示之前是 U 模式，否則表示 S 模式。如果 SPP 是 1 那就出現了嚴重錯誤（因為既然是從 U 切換到 S 的過程，怎麼可以 SPP 是 S 模式呢？當然，如果是核心執行時發生了中斷 SPP 是 1 那自然是對的，核心執行時發生中斷時如果檢查 SPP 是 0 那也是嚴重的錯誤）。

**S 切換到 U**[**#**](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#1009804242)

在從 S 切換到 U 時，要手動清除 sstatus 的 SPP 欄位，將其置為零；將 sstatus 的 SPIE 欄位置為 1，啟用使用者中斷；設定 sepc 為使用者處理程序的 PC 值（你可能疑惑在 U 轉換到 S 時不是已經將使用者處理程序的保存在了 sepc 了嗎?因為在 S-mode 也會發生中斷呀，那麼 sepc 就會被用來保存發生中斷位置時的 PC 了）。如果啟用了頁表，就需要想還原使用者處理程序的頁表，即將使用者處理程序的頁表地址寫入 satp，之後恢復上下文，然後執行 sret 指令，硬體會自動完成以下操作：

1. 從 sepc 暫存器中取出要恢復的下一條指令地址，將其複製到程序計數器 pc 中，以恢復現場；
2. 從 sstatus 暫存器中取出使用者模式的相關狀態，包括中斷使能位、虛擬儲存模式等，以恢復使用者模式的狀態；
3. 將當前特權模式設定為使用者模式，即取消特權模式，回到使用者模式。

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**S 與 M 之間的切換[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "4272736558)**

**S 切換到 M**[**#**](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#3620055739)

S 切換到 M 與從 U 切換到 M 類似，都是從低特權級到高特權級的切換。在 S 運行的程式碼，也可以通過 ecall 指令陷入到 M 中。

1. S-mode 的程式碼執行一個指令觸發了異常或陷阱，例如環境呼叫（ECALL）指令
2. 處理器將當前的 S-mode 上下文的狀態保存下來，包括程序計數器 (PC)、S-mode 特權等級和其他相關暫存器，保存在當前特權等級堆疊中的 S-MODE 陷阱幀（trap frame，其實就是一個頁面）中
3. 處理器通過將 mstatus 暫存器中的 MPP 欄位設定為 0b11（表示先前的模式是 S 模式）將特權等級設定為 M-mode
4. 處理器將程序計數器設定為在 M-mode 中的陷阱處理程序例程的地址
5. 處理器還在 mstatus 暫存器中設定 M-mode 中斷使能位 (MIE) 為 0，以在陷阱處理程序中停用中斷

**系統呼叫的實現**

系統呼叫是利用異常機制實現的。在 mcause 中我們看到有 Environment call from U-mode 和 Environment call from S-mode 兩個異常類型。那麼如何觸發這兩個異常呢？分別在 U-mode 和 S-mode 執行 ecall 指令就能觸發這兩個異常了。

異常觸發之後，就會被捕捉到 M-mode（我之前提過，RISC v 下默認是把所有的異常、中斷捕捉到 M-mode，當且僅當對應的陷阱被委託給了其它模式才會陷入到被委託的模式中）。假如說

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**地址空間佈局[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "2889809562)**

啟用分頁模式下，核心程式碼的訪存地址也會被視為一個虛擬地址並需要經過 MMU 的地址轉換，因此我們也需要為核心對應構造一個地址空間，它除了仍然需要允許核心的各資料段能夠被正常訪問之後，還需要包含所有應用的核心棧以及一個 跳板 (Trampoline) 。

值得注意的是，下面是是 rCore 的核心地址空間分佈，不同的 OS 設計不同。

|  |
| --- |
| **高 256GB 核心地址空間** |

|  |
| --- |
| **低 256GB 核心地址空間**  [一張含有 圖表 的圖片  自動產生的描述](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210015309-973629164.png) |

|  |
| --- |
| **應用程式高 256GB 地址空間** |

|  |
| --- |
| **應用程式低 256GB 地址空間**  [一張含有 圖表 的圖片  自動產生的描述](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210016165-419532978.png) |

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**跳板機制[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "1152981011)**

使能了分頁機制之後，我們必須在 trap 過程中同時完成地址空間的切換。具體來說，當 \_\_alltraps 保存 Trap 上下文的時候，我們必須通過修改 satp 從應用地址空間切換到核心地址空間，因為 trap handler 只有在核心地址空間中才能訪問；同理，在 \_\_restore 恢復 Trap 上下文的時候，我們也必須從核心地址空間切換回應用地址空間，因為應用的程式碼和資料只能在它自己的地址空間中才能訪問，應用是看不到核心地址空間的。這樣就要求地址空間的切換不能影響指令的連續執行，即要求應用和核心地址空間在切換地址空間指令附\*是\*滑的。

我們為何將應用的 Trap 上下文放到應用地址空間的次高頁面而不是核心地址空間中的核心棧中呢？原因在於，在保存 Trap 上下文到核心棧中之前，我們必須完成兩項工作：1）必須先切換到核心地址空間，這就需要將核心地址空間的 頁表地址寫入 satp 暫存器；2）之後還需要保存應用的核心棧棧頂的位置，這樣才能以它為基址保存 Trap 上下文。這兩步需要用暫存器作為臨時周轉，然而我們無法在不破壞任何一個通用暫存器的情況下做到這一點。因為事實上我們需要用到核心的兩條資訊：核心地址空間的 頁表地址，以及應用的核心棧棧頂的位置，RISC-V 卻只提供一個 sscratch 暫存器可用來進行周轉。所以，我們不得不將 Trap 上下文保存在應用地址空間的一個虛擬頁面中，而不是切換到核心地址空間去保存。

**Page fault**

當 CPU 無法將虛擬地址轉換為實體位址時，CPU 會生成頁面錯誤異常。RISC-V 有三種不同類型的頁面錯誤：載入頁面錯誤(當載入指令無法轉換其虛擬地址時)、儲存頁面錯誤(當儲存指令無法轉換其虛擬地址時)和指令頁面錯誤(當指令的地址不轉換時)。scause 暫存器中的值指示頁面錯誤的類型，而 stval 暫存器中包含無法轉換的地址。

Cow(copy on write) Fork 中的基本方案是讓父子處理程序在最開始時共享所有物理頁面，但將它們對應為唯讀。因此，當子處理程序或父處理程序執行儲存指令時，RISC-V CPU 會引發頁面錯誤異常。作為對此異常的響應，核心會複製包含錯誤地址的頁面。它將一個副本對應到子處理程序的地址空間中，並將另一個副本對應到父處理程序的地址空間中。在更新頁表之後，核心在導致錯誤的指令處恢復出錯處理程序。因為核心已經更新了相關的 PTE 以允許寫入，所以出錯指令現在將正常執行。

**xv6 中是如何設定 stvec 的**

我們已經知道 stvec 暫存器保存的是中斷服務程序的首地址，另外在 U 模式下，stvec 必須指向的是 uservec，在 S 模式下，stvec 必須指向的是 kernelvec，這樣做的原因是需要在 uservec 切換頁表。

那麼 xv6 是如何設定 stvec 的呢？首先在 uservec 例程中除了執行保存上下文、切換頁表等操作之外，還會在 usertrap 中將 stvec 指向 kernelvec，這裡的切換的目的是當前已經執行到了 S 模式，所有的中斷、陷阱等都必須由 kernelvec 負責處理。

當需要返回 usertrap 時，usertrap 會呼叫 usertrapret，usertrapret 會重新設定 stvec 的值使其指向 uservec，之後跳轉到 userret，恢復上下文和切換頁表。

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**第一次的 stvec 是如何設定的[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "1562710163)**

在 main 中，cpu0 呼叫了 userinit() 建立了第一個使用者處理程序，並在 scheduler 中會切換到該處理程序。該處理程序的上下文中的 ra(返回地址)被設定成了 forkret()，當 scheduler 執行 swtch 函數時，會將處理程序上下文中的 ra 寫入到 ra 暫存器中，這樣當要從 swtch() 中返回時，就會返回到了 forkret()，在 forkret() 中會直接呼叫 usertrapret 以實現 stvec 的設定和頁表的切換。

**與中斷有關的硬體單元**

在 RISC v 中，與中斷有關的硬體單元主要有 [ACLINT](https://github.com/riscv/riscv-aclint/blob/main/riscv-aclint.adoc)、CLINT、[PLIC](https://github.com/riscv/riscv-plic-spec/blob/master/riscv-plic.adoc" \t "_blank)、[CLIC](https://github.com/riscv/riscv-fast-interrupt/blob/master/clic.adoc" \t "_blank)。

CLINT 的全稱是 Core Local Interrupt，ACLINT 的全稱是 Advanced Core Local Interrupt, CLIC 的全稱是 Core-Local Interrupt Controller。

PLIC 的全稱 Platform-Level Interrupt Controller。

儘管 CLIC 與 PLIC 名稱相似，但是 CLIC 其實是為取代 CLINT 而設計的。ACLINT 是為了取代 SiFive CLINT 而設計的，本質上講，ACLINT 相比於 CLINT 的優勢就在於進行了模組化設計，將定時器和 IPI 功能分開了，同時能夠支援 NUMA 系統。但是 ACLINT 和 CLINT 都還是 RISC-V basic local Interrupts 的範疇。

PLIC 和 CLIC 的區別在於，前者負責的是整個\*台的外部中斷，CLIC 負責的是每個 HART 的本地中斷。

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**PLIC**[**#**](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#1738934816)

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**ACLINT**[**#**](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#4263514564)

ACLINT 的規範翻譯參見 [RISC-V ACLIT](https://wbc3ji2vof.feishu.cn/wiki/wikcnLbXPxSlB2x1kCV3jU2WiXf)

根據 [Linux RISC-V ACLINT Support](https://lore.kernel.org/lkml/20211007123632.697666-1-anup.patel@wdc.com/) 的說法，大多數現有的 RISC-V \*台使用 SiFive CLINT 來提供 M 級定時器和 IPI 支援，而 S 級使用 SBI 呼叫定時器和 IPI。此外，SiFive CLINT 裝置是一個單一的裝置，所以 RISC-V \*台不能部分實現提供定時器和 IPI 的替代機制。RISC-V 高級核心本地中斷器(ACLINT)嘗試通過以下方式解決 SiFive CLINT 的限制:

1. 採用模組化方法，分離定時器和 IPI 功能為不同的裝置，以便 RISC-V \*台可以只包括所需的裝置
2. 為 S 級 IPI 提供專用的 MMIO 裝置，以便 SBI 呼叫可以避免在 Linux RISC-V 中使用 IPI
3. 允許定時器和 IPI 裝置的多個實例多 sockets NUMA 系統

RISC-V ACLINT 規範向後相容 SiFive CLINT。

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**CLIC**[**#**](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#150717829)

spec 參見 [riscv-fast-interrupt/clic.adoc](https://github.com/riscv/riscv-fast-interrupt/blob/master/clic.adoc" \l "background-and-motivation" \t "_blank)

RISC-V 特權架構規範定義了 CSR，例如 x ip 、 x ie 和中斷行為。為這種 RISC-V 中斷方案提供處理器間中斷和定時器功能的簡單中斷控製器被稱為 CLINT。當 x tvec .mode 設定為 00 或 01 時，本規範將使用術語 CLINT 模式。

在前文介紹 mtvec 時提到了 mode 欄位，在 RISC-V 目前的特權級規範中，mode 欄位只能取 00 或 01，其它值是 reserved。從 spec 的描述中我們可以看出，mode 欄位無論是 00 還是 01，都是 CLINT 模式，因此我們在前文介紹的有關中斷的介紹都是 CLINT 模式（包括 ACLINT）。

我目前不太清除 CLIC 是否在

**時鐘中斷**

“定時器中斷”是由一個獨立的計時器電路發出的訊號，表示預定的時間間隔已經結束。計時器子系統將中斷當前正在執行的程式碼。定時器中斷可以由作業系統處理，用於實現時間片多執行緒，但是對於 MTIME 和 MTIMECMP 的讀寫只能由 M-mode 的程式碼實現，因此核心需要呼叫 SBI 的服務。

我相信你已經在 [RISC-V ACLIT](https://wbc3ji2vof.feishu.cn/wiki/wikcnLbXPxSlB2x1kCV3jU2WiXf) 已經瞭解到了時鐘中斷的基本原理，現在我們看一下如何處理時鐘中斷。

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**時鐘中斷相關的暫存器[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "3018343341)**

<https://tinylab.org/riscv-timer/>  
mtime 需要以固定的頻率遞增，並在發生溢出時迴繞。當 mtime 大於或等於 mtimecmp 時，由核內中斷控製器 (CLINT, Core-Local Interrupt Controller) 產生 timer 中斷。中斷的使能由 mie 暫存器中的 MTIE 和 STIE 位控制，mip 中的 MPIE 和 SPIE 則指示了 timer 中斷是否處於 pending。在 RV32 中讀取 mtimecmp 結果為低 32 位， mtimecmp 的高 32 位需要讀取 mtimecmph 得到。  
由於 mtimecmp 只能在 M 模式下訪問，對於 S/HS 模式下的核心和 VU/VS 模式下的虛擬機器需要通過 SBI 才能訪問，會造成較大的中斷延遲和性能開銷。為瞭解決這一問題，RISC-V 新增了 Sstc 拓展支援（已批准但尚未最終整合到規範中）。  
[Sstc 擴展](https://github.com/riscv/riscv-time-compare)為 HS 模式和 VS 模式分別新增了 stimecmp 和 vstimecmp 暫存器，當time>=stimecmp或者time + htimedelta>=vstimecmp是會產生 timer 中斷，不再需要通過 SBI 陷入到其它模式。

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**時鐘中斷的基本處理過程[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "2696113960)**

如下圖所示是時鐘中斷的基本過程(xv6 的處理過程)：

[一張含有 圖表 的圖片

自動產生的描述](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210016664-572388837.png)

圖源：<https://shakti.org.in/docs/risc-v-asm-manual.pdf>

讓我們首先回顧一下有關 timer 的暫存器。首先要明確的是，timer 的暫存器在 timer 裝置裡，不在 CPU 中，是通過 MMIO 的方式對應到記憶體中的。

mtime 暫存器是一個同步計數器。它從處理器上電開始運行，並以 tick 單位提供當前的即時時間。

mtimecmp 暫存器用於儲存定時器中斷應該發生的時間間隔。mtimecmp 的值與 mtime 暫存器進行比較。當 mtime 值變得大於 mtimecmp 時，就會產生一個定時器中斷。mtime 和 mtimecmp 暫存器都是 64 位記憶體對應暫存器，因此可以直接按照記憶體讀寫的方式修改這兩個暫存器的值。

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**xv6 的實現[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "1111323725)**

xv6 對於時鐘中斷的處理方式是這樣的：在 M-mode 設定好時鐘中斷的處理函數，當發生時鐘中斷時就由 M-mode 的程式碼讀寫 mtime 和 mtimecmp，然後啟動 sip.SSIP 以軟體中斷的形式通知核心。核心在收到軟體中斷之後會遞增 ticks 變數，並呼叫 wakeup 函數喚醒沉睡的處理程序。 核心本身也會收到時鐘中斷，此時核心會判斷當前運行的是不是處理程序號為 0 的處理程序，如果不是就會呼叫 yield() 函數使當前處理程序放棄 CPU 並調度下一個處理程序；如果使處理程序號為 0 的處理程序，那就不做處理。

**timer\_init[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "2121756663)**

|  |
| --- |
| // core local interruptor (CLINT), which contains the timer.  #define CLINT 0x2000000L  #define CLINT\_MTIMECMP(hartid) (CLINT + 0x4000 + 8\*(hartid))  #define CLINT\_MTIME (CLINT + 0xBFF8) // cycles since boot.  void  timerinit()  {  // each CPU has a separate source of timer interrupts.  int id = r\_mhartid();  // ask the CLINT for a timer interrupt.  int interval = 1000000; // cycles; about 1/10th second in qemu.    // 我已經提過，mtimecmp 是對應到了實體位址中的，因此可以直接按照記憶體讀寫的方式  // 修改暫存器的值  // MTIME 暫存器對應到了 0x2000\_BFF8  // 一塊CPU有一個MTIME，所有的hart都共用這一個 MTIME  // MTIMECMP 的記憶體基地址是 0x2000000L  // 每個暫存器佔 8個位元組，每個hart都有一個MTIMECMP暫存器  // 因此呢，第id個（從0開始計數）的hart對應的 MTIMECMP 的暫存器的實體位址就是  // 0x2000000L + 8 \* id  // 因此呢就容易理解下面的操作了，實際上就是根據 MTIME 初始化 MTIMECMP  \*(uint64\*)CLINT\_MTIMECMP(id) = \*(uint64\*)CLINT\_MTIME + interval;  // prepare information in scratch[] for timervec.  // scratch[0..2] : space for timervec to save registers.  // scratch[3] : address of CLINT MTIMECMP register.  // scratch[4] : desired interval (in cycles) between timer interrupts.  uint64 \*scratch = &timer\_scratch[id][0];  scratch[3] = CLINT\_MTIMECMP(id);//記錄當前hart對應的 MTIMECMP 暫存器對應到的實體位址  scratch[4] = interval;  w\_mscratch((uint64)scratch);//將陣列指針寫入mscratch  // set the machine-mode trap handler.  w\_mtvec((uint64)timervec);  // enable machine-mode interrupts.  w\_mstatus(r\_mstatus() | MSTATUS\_MIE);  // enable machine-mode timer interrupts.  w\_mie(r\_mie() | MIE\_MTIE);  } |

**時鐘中斷處理函數[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "1930270006)**

在下面的程式碼中，首先是將 mscratch 與 a0 暫存器交換了值，此時 a0 保存的值就是個陣列指針(這一點在前面的 timer\_init 中已經分析了)。

timervec:

|  |
| --- |
| # start.c has set up the memory that mscratch points to:  # scratch[0,8,16] : register save area.  # scratch[24] : address of CLINT's MTIMECMP register.  # scratch[32] : desired interval between interrupts.    csrrw a0, mscratch, a0    # 保存暫存器的上下文  sd a1, 0(a0)  sd a2, 8(a0)  sd a3, 16(a0)  # schedule the next timer interrupt  # by adding interval to mtimecmp.  # 實際上執行的就是 MTIMECMP = MTIME + INTERVAL  ld a1, 24(a0) # CLINT\_MTIMECMP(hart)  ld a2, 32(a0) # interval  ld a3, 0(a1)  add a3, a3, a2  sd a3, 0(a1)  # arrange for a supervisor software interrupt  # after this handler returns.  # 通過supervisor software 中斷的方式通知 S-mode 的核心處理時鐘中斷  # 實際上呢，時鐘中斷已經在M-mode被處理掉了  # 之所以還要通知S-mode的核心是因為核心的處理程序調度器依賴於對時間的掌握  # S-mode只是根據時鐘變化去做處理程序調度器相關的處理  li a1, 2  csrw sip, a1  # 恢復上下文  ld a3, 16(a0)  ld a2, 8(a0)  ld a1, 0(a0)  csrrw a0, mscratch, a0  mret |

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**Linux 的時鐘中斷的實現[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "427107886)**

參見 [RISC-V timer 在 Linux 中的實現 - 泰曉科技](https://tinylab.org/riscv-timer/)

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**QEMU 的時鐘中斷的邏輯[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "2654002713)**

參見 <https://wangzhou.github.io/riscv-timer%E7%9A%84%E5%9F%BA%E6%9C%AC%E9%80%BB%E8%BE%91/>

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**參考文獻**[**#**](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#1763661800)

* [wangzhou.github.io](https://wangzhou.github.io/riscv-timer%E7%9A%84%E5%9F%BA%E6%9C%AC%E9%80%BB%E8%BE%91/)
* [RISC-V timer 在 Linux 中的實現 - 泰曉科技](https://tinylab.org/riscv-timer/)
* <https://shakti.org.in/docs/risc-v-asm-manual.pdf>
* RISC-V ACLINT Spec
* RISC-V Privileged Spec

**軟體中斷**

所謂軟體中斷就是軟體觸發的中斷，也是所謂的核間中斷（inter-process interrupt，IPI）。在 RISC v 中，核間中斷是通過設定 MIP 的 MSIP 或者 SSIP 實現的。

下面以 Linux 和 opensbi 為例介紹 S-MODE 的軟體中斷的實現。

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**中斷發送**[**#**](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#4286954871)

**Linux 核心實現[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "2833633560)**

在 arch/riscv/kernel/smp.c 中實現了 ipi 傳送和處理的若干函數。

首先應當明確的是，IPI 是核間中斷，也就是一個核向另一個核傳送的中斷，那麼就是軟體執行階段出於某種目的向另一個/些核傳送了中斷，那麼就需要告知這個/些核，讓這些核做某些事情，這就需要向其它核傳送消息。

在 smp.c 中定義了列舉值：

|  |
| --- |
| enum ipi\_message\_type {  IPI\_RESCHEDULE,  IPI\_CALL\_FUNC,  IPI\_CPU\_STOP,  IPI\_IRQ\_WORK,  IPI\_TIMER,  IPI\_MAX  }; |

從這些列舉值我們可以看出，一個軟體中斷可以傳遞 5 種不同的中斷消息。

這些消息需要保存在變數裡，因此在 smp.c 中也定義了靜態變數 ipi\_data：

首先看靜態變數 ipi\_data，該變數定義如下：

|  |
| --- |
| static struct {  unsigned long stats[IPI\_MAX] \_\_\_\_cacheline\_aligned;//記錄對應類型的IPI收到了多少個  unsigned long bits \_\_\_\_cacheline\_aligned;//記錄對應的IPI是否被啟動  } ipi\_data[NR\_CPUS] \_\_cacheline\_aligned; |

從定義中我們可以看出，每個 HART 都有一個獨立的 ipi\_data 且是快取行對齊的。其中 stats 記錄了傳送的軟體中斷的所傳遞的消息。在傳送 IPI 之前，當前核心需要將資訊寫入到 ipi\_data 變數中，這樣當其它核心收到 IPI 並處理時，就可以根據 ipi\_data 中記錄的值進行相關操作。

這裡我以向單個核傳送 IPI 為例進行介紹：

|  |
| --- |
| static void send\_ipi\_single(int cpu, enum ipi\_message\_type op)  {  smp\_mb\_\_before\_atomic();  set\_bit(op, &ipi\_data[cpu].bits);  smp\_mb\_\_after\_atomic();  if (ipi\_ops && ipi\_ops->ipi\_inject)  ipi\_ops->ipi\_inject(cpumask\_of(cpu));  else  pr\_warn("SMP: IPI inject method not available\n");  } |

我們可以看到兩個參數，第一個參數 cpu 是要傳送到哪個核心的編號，op 則是要傳遞的 IPI 類型。

set\_bit 就是啟動對應的 IPI 類型。

這裡比較關鍵的是呼叫了 ipi\_inject，這是個函數指針，該函數指針指向了 sbi\_send\_cpumask\_ipi 函數。

[一張含有 文字 的圖片

自動產生的描述](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210017032-67385686.png)

在 arch/riscv/kernel/sbi.c 中，我們看到 sbi\_send\_cpumask\_ipi 也是一個函數指針，它的實現實際上與 sbi 的標準有關，比如有 \_\_sbi\_send\_ipi\_v01，\_\_sbi\_send\_ipi\_v02 等函數。

無論是哪種規範吧，反正最終是呼叫到了 sbi，下面我們以 opensbi 為例繼續介紹軟體中斷的過程。

**Opensbi[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "1523663380)**

在 opensbi/lib/sbi/sbi\_ipi.c 中實現了 ipi send 的相關函數。

[一張含有 文字 的圖片

自動產生的描述](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210017553-311422841.png)

從呼叫函數棧中，可以看出，最終呼叫到了 mswi\_ipi\_send 函數：

|  |
| --- |
| static void mswi\_ipi\_send(u32 target\_hart)  {  u32 \*msip;  struct aclint\_mswi\_data \*mswi;  if (SBI\_HARTMASK\_MAX\_BITS <= target\_hart)  return;  mswi = mswi\_hartid2data[target\_hart];  if (!mswi)  return;  /\* Set ACLINT IPI \*/  msip = (void \*)mswi->addr;  writel(1, &msip[target\_hart - mswi->first\_hartid]);  } |

通過將 CSR\_MIP.SSIP 置為就實現了 S-MODE 軟體中斷，因為根據 RISC v 的中斷委託機制，中斷會最終拉高 CSR\_SIP.SSIP，並在 S-MODE 對軟體中斷進行處理。下面我們來看 Linux 是如何對軟體中斷進行處理的。

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**中斷處理[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "2051089585)**

S-MODE 的軟體中斷處理自然在 Linux 核心中。在 arch/riscv/kernel/smp.c 的 handle\_IPI 函數就是軟體中斷處理函數。

|  |
| --- |
| void handle\_IPI(struct pt\_regs \*regs)  {  unsigned long \*pending\_ipis = &ipi\_data[smp\_processor\_id()].bits;  unsigned long \*stats = ipi\_data[smp\_processor\_id()].stats;  riscv\_clear\_ipi();//這裡並不會丟失IPI，因為IPI傳送的數量和啟動狀態已經記錄在了ipi\_data裡面  // 下面就是對ipi的具體處理嘍，讀者有興趣可自行查看  while (true) {  unsigned long ops;  /\* Order bit clearing and data access. \*/  mb();  ops = xchg(pending\_ipis, 0);  if (ops == 0)  return;  if (ops & (1 << IPI\_RESCHEDULE)) {  stats[IPI\_RESCHEDULE]++;  scheduler\_ipi();  }  if (ops & (1 << IPI\_CALL\_FUNC)) {  stats[IPI\_CALL\_FUNC]++;  generic\_smp\_call\_function\_interrupt();  }  if (ops & (1 << IPI\_CPU\_STOP)) {  stats[IPI\_CPU\_STOP]++;  ipi\_stop();  }  if (ops & (1 << IPI\_IRQ\_WORK)) {  stats[IPI\_IRQ\_WORK]++;  irq\_work\_run();  }  #ifdef CONFIG\_GENERIC\_CLOCKEVENTS\_BROADCAST  if (ops & (1 << IPI\_TIMER)) {  stats[IPI\_TIMER]++;  tick\_receive\_broadcast();  }  #endif  BUG\_ON((ops >> IPI\_MAX) != 0);  /\* Order data access and bit testing. \*/  mb();  }  } |

**User-Level interrupt**

上一節敘述的是在 M-S-U 的 CPU 中的標準中斷，這一節描述使用者態中斷。

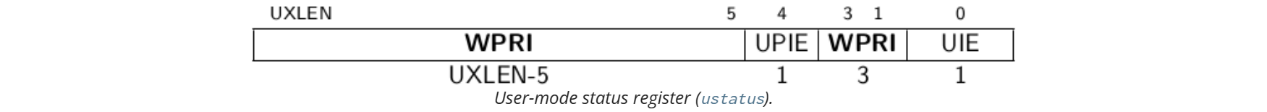
使用者態中斷是 N Standard Extension，相關實現可以參考 <https://github.com/TRCYX/riscv-user-level-interrupt> 和 <https://gallium70.github.io/rv-n-ext-impl/ch1_1_priv_and_trap.html>

事實上使用者態中斷比較罕見，但是 x86 已經完全支援使用者態中斷了。

與使用者態中斷有關的暫存器有：ustatus, uip, uie, sedeleg, sideleg, uscratch, uepc, utevc, utval。其中 sedeleg 和 sideleg 就是為實現使用者態中斷而新增的，如果 S-mode 不委託異常、中斷到 U-mode，那麼使用者態中斷是無法實現的。sedeleg/sideleg 與 medeleg/mideleg 是完全一致的，不贅述。

uscratch/uepc/utevc/utval 與相應的 M-mode 的暫存器也是一致的，不再贅述。這裡僅重點介紹 ustatus, uip, uie。

**ustatus[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "4119110290)**

[](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210018160-1168655986.png)

ustatus 是很簡單的，就兩個值得注意的欄位 UPIE 和 UIE。如果 UIE 為 0 就停用使用者態中斷，否則啟用使用者態中斷。在處理使用者態中斷時，使用 UPIE 記錄 UIE，之後會將 UIE 置零。值得注意的是，ustatus 裡面沒有 UPP，因為沒有比 U-mode 更低的特權級了，陷入到 U-mode 的一定是 U-mode 的特權級，因此也就沒有必要記錄發生中斷前的特權級了。

**uip 與 uie[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "1910827469)**

[一張含有 資料表 的圖片

自動產生的描述](https://img2023.cnblogs.com/blog/1653979/202307/1653979-20230712210018778-56903842.png)

本規範定義了三種中斷類型：軟體中斷、定時器中斷和外部中斷。可以通過向 uip 暫存器的使用者軟體中斷掛起（USIP）位寫入 1，來觸發當前處理器上的使用者級軟體中斷。可以通過向 uip 暫存器的 USIP 位寫入 0，來清除掛起的使用者級軟體中斷。當 uie 暫存器中的 USIE 位清零時，使用者級軟體中斷將被停用。

ABI 應該提供一種機制，以傳送處理器間中斷到其他處理器，從而最終導致接收處理器的 uip 暫存器中的 USIP 位被設定。

除了 uip 暫存器中的 USIP 位之外，其餘所有位都是唯讀的。

如果 uip 暫存器中的 UTIP 位被設定，則表示使用者級定時器中斷掛起。當 uie 暫存器中的 UTIE 位清零時，將停用使用者級定時器中斷。ABI 應該提供一種機制來清除掛起的定時器中斷。

如果 uip 暫存器中的 UEIP 位被設定，則表示使用者級外部中斷掛起。當 uie 暫存器中的 UEIE 位清零時，將停用使用者級外部中斷。ABI 應該提供一些方法來遮蔽、解除遮蔽和查詢外部中斷的原因。

uip 和 uie 暫存器是 mip 和 mie 暫存器的子集。讀取 uip/uie 的任何欄位或寫入其任何可寫字段，都會導致 mip/mie 中同名欄位的讀寫。如果實現了 S 模式，則 uip 和 uie 暫存器也是 sip 和 sie 暫存器的子集。

[回到頂部](https://www.cnblogs.com/harrypotterjackson/p/17548837.html#_labelTop)

**參考資料[#](https://www.cnblogs.com/harrypotterjackson/p/17548837.html" \l "1218014258)**

* [User Interrupt](https://0x10.sh/user-interrupt)

分類: [作業系統程式設計](https://www.cnblogs.com/harrypotterjackson/category/1757421.html) , [Linux](https://www.cnblogs.com/harrypotterjackson/category/1757423.html) , [分佈式系統](https://www.cnblogs.com/harrypotterjackson/category/2043779.html)

標籤: [riscv](https://www.cnblogs.com/harrypotterjackson/tag/riscv/) , [中斷](https://www.cnblogs.com/harrypotterjackson/tag/%E4%B8%AD%E6%96%AD/)