微自幹的作業系統輕旅行

Ian Chen

About me

- NYCU CS
- Google DSC Lead @ NCTU
- COSCUP 2021 IT Team Lead
- 社群分享
    - SITCON
    - COSCUP
    - 六角學院
    - 快樂學程式

Preliminary work

- 重新學習 C 語言
- 學習 System Programing (CS241@ Illinois)
- RISC-V Spec
- RISC-V 架構與嵌入式開發快速入門
- OS 恐龍本
- Trace 相關專案的原始碼

Google Developer Groups

Google Developer Groups

Outline

The Skills you got to know

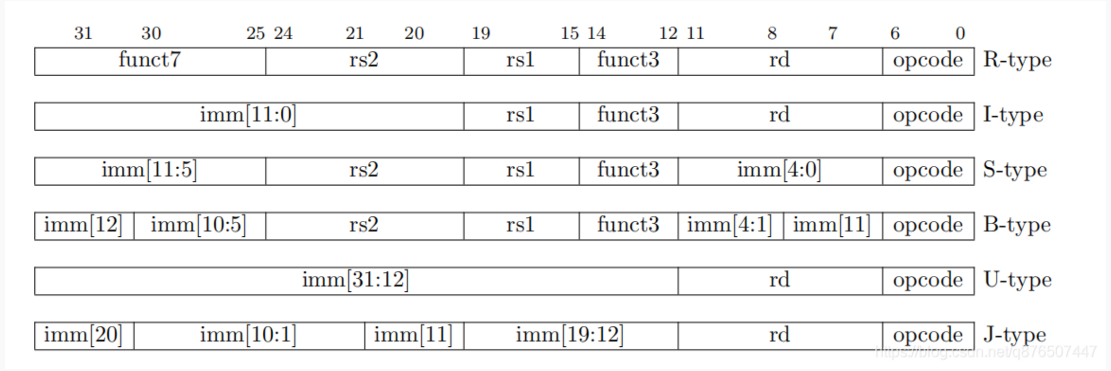
1. Virtual Machine or Evaluation Board
2. RISC-V GNU Compiler Toolchain
3. Git
4. C Programming Skill
5. Assembly code

The History of RISC-V

- 由 UC Berkeley 發起
- 開源硬體
- 有前途
- 怎麼會接觸到它?

Google Developer Groups

Instruction Set - RV32I

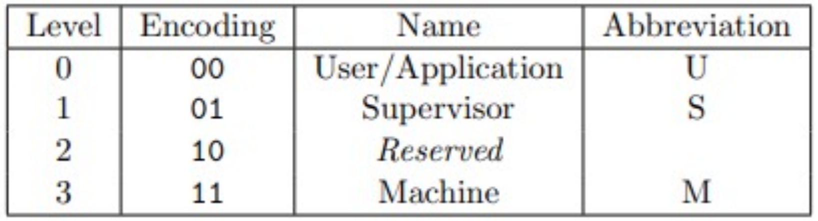


31	30 … 25	24 … 21	20	19 … 15	14 … 12	11 … 8	7	6 … 0	
funct7		rs2		rs1	funct3	rd		opcode	R-type
imm[11:0]				rs1	funct3	rd		opcode	I-type
imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode	S-type
imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
imm[31:12]						rd		opcode	U-type
imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode	J-type

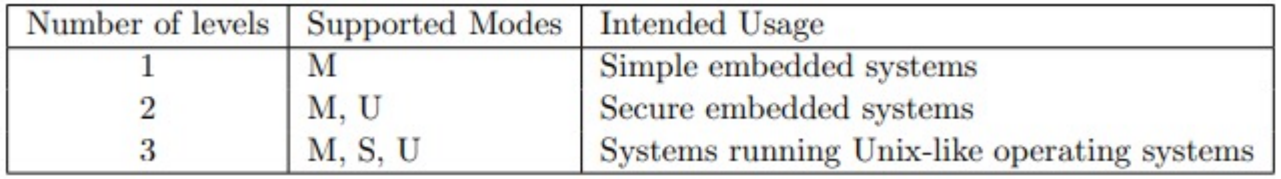
- Bit 20 - Bit 31: 用來表示立即數 (立即尋址方式指令中給出的數)
- Bit 15 - Bit 19: 表示 rs1 (Input 暫存器)
- Bit 12 - Bit 14: 表示 funct (確定 Type 後，識別指令用)
- Bit 7 - Bit 11: 表示 rd (目標暫存器)
- Bit 0 - Bit 6: 為 opcode (用於辨別 Type)

Privileged architecture

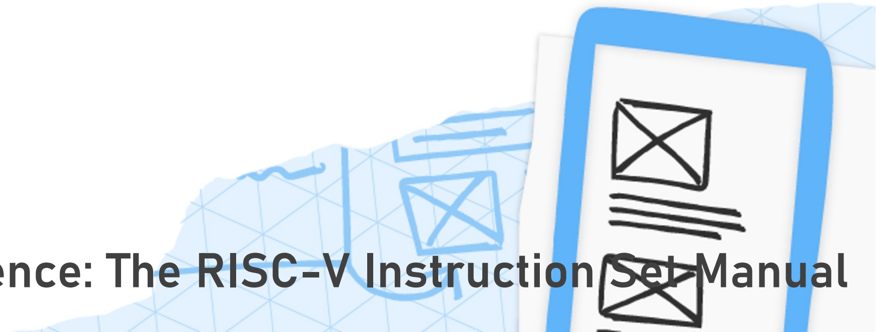
- U/S/M Mode 各自持有一組 CSR 暫存器
- 不同的特權模式對應到不同的執行權限
    - M mode 通常用來執行 Trusted Program
    - S mode 執行 OS Kernel
    - U mode 執行 User Application
- 本議程介紹的專案僅用到 Machine Mode



Level	Encoding	Name	Abbreviation
0	00	User/Application	U
1	01	Supervisor	S
2	10	*Reserved*	
3	11	Machine	M



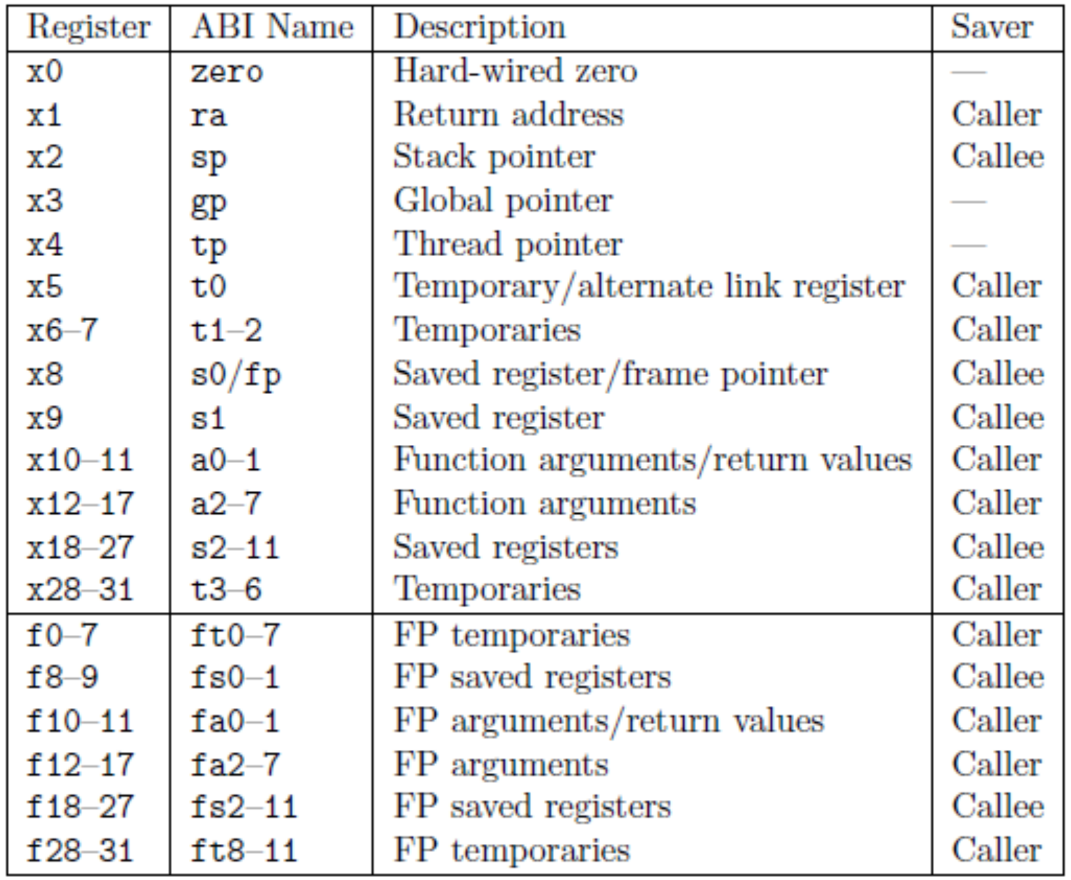
Number of levels	Supported Modes	Intended Usage
1	M	Simple embedded systems
2	M, U	Secure embedded systems
3	M, S, U	Systems running Unix-like operating systems



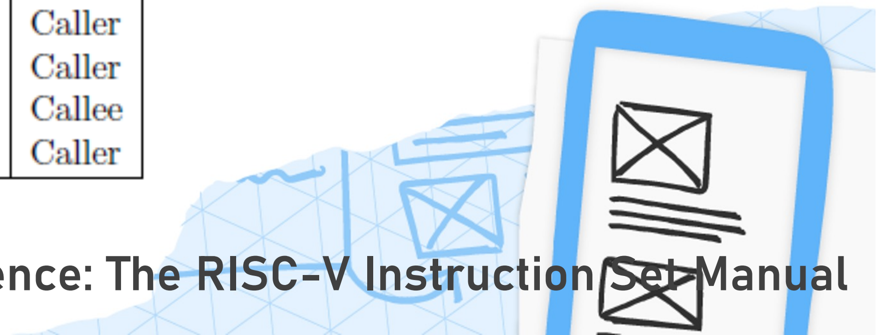
Reference: The RISC-V Instruction Set Manual

Calling Convention

- Caller save?
- Callee save?



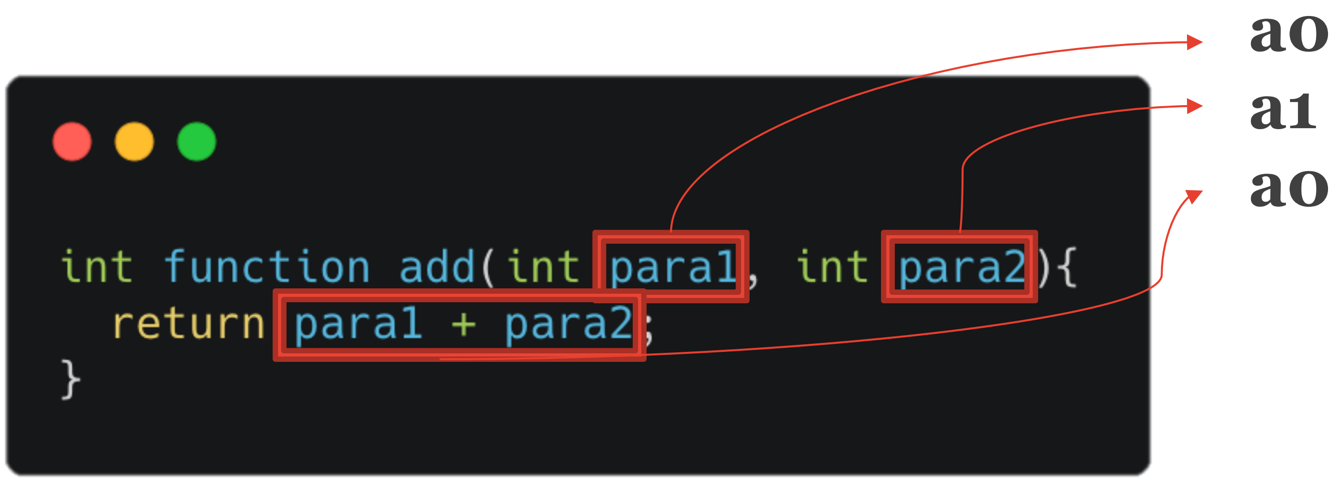
Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller



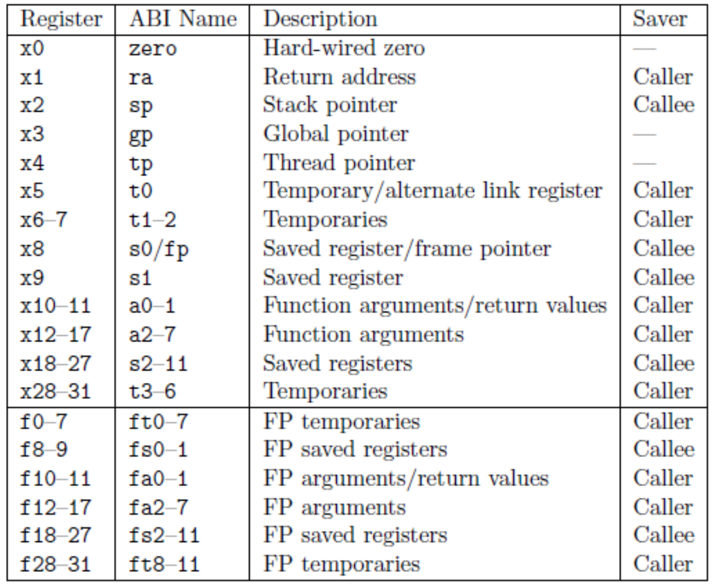
Google Developer Groups

Reference: The RISC-V Instruction Set Manual

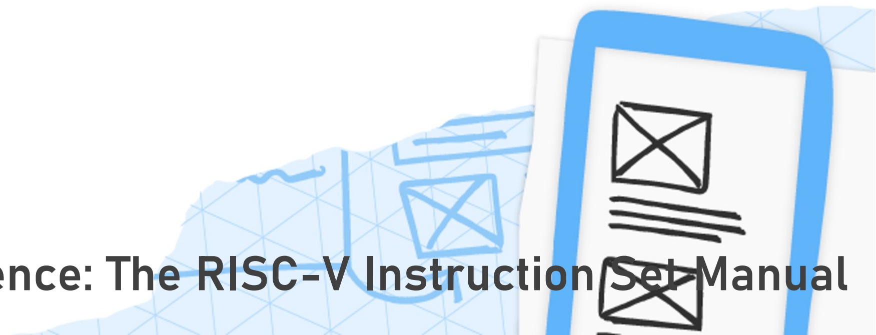
Calling Convention



```
int function add(int para1, int para2){
    return para1 + para2;
}
```



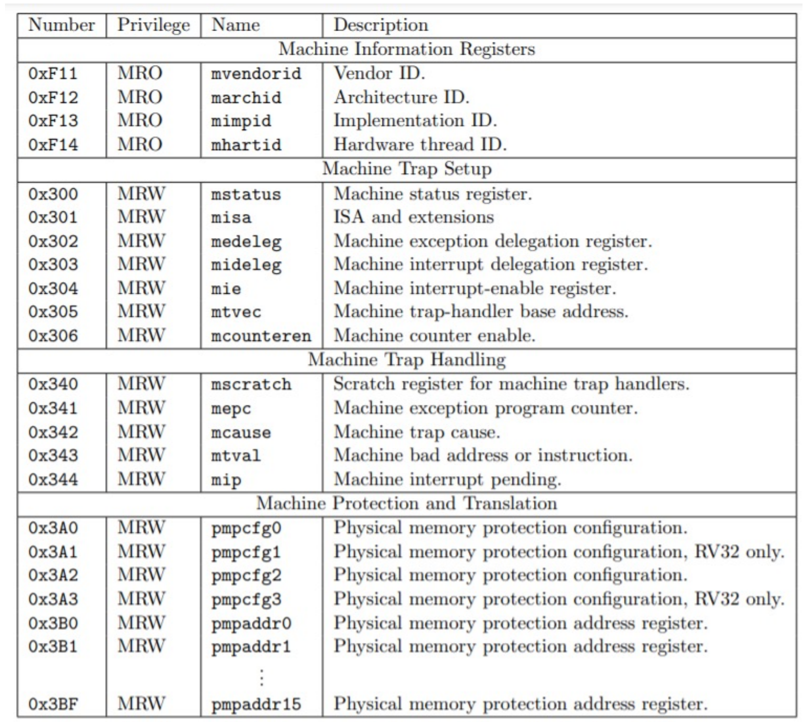
Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller



Google Developer Groups

Reference: The RISC-V Instruction Set Manual

Control and Status Register



Number	Privilege	Name	Description
			Machine Information Registers
0xF11	MRO	mvendorid	Vendor ID.
0xF12	MRO	marchid	Architecture ID.
0xF13	MRO	mimpid	Implementation ID.
0xF14	MRO	mhartid	Hardware thread ID.
			Machine Trap Setup
0x300	MRW	mstatus	Machine status register.
0x301	MRW	misa	ISA and extensions
0x302	MRW	medeleg	Machine exception delegation register.
0x303	MRW	mideleg	Machine interrupt delegation register.
0x304	MRW	mie	Machine interrupt-enable register.
0x305	MRW	mtvec	Machine trap-handler base address.
0x306	MRW	mcounteren	Machine counter enable.
			Machine Trap Handling
0x340	MRW	mscratch	Scratch register for machine trap handlers.
0x341	MRW	mepc	Machine exception program counter.
0x342	MRW	mcause	Machine trap cause.
0x343	MRW	mtval	Machine bad address or instruction.
0x344	MRW	mip	Machine interrupt pending.
			Machine Protection and Translation
0x3A0	MRW	pmpcfg0	Physical memory protection configuration.
0x3A1	MRW	pmpcfg1	Physical memory protection configuration, RV32 only.
0x3A2	MRW	pmpcfg2	Physical memory protection configuration.
0x3A3	MRW	pmpcfg3	Physical memory protection configuration, RV32 only.
0x3B0	MRW	pmpaddr0	Physical memory protection address register.
0x3B1	MRW	pmpaddr1	Physical memory protection address register.
		⋮	
0x3BF	MRW	pmpaddr15	Physical memory protection address register.

- U/S/M Mode 各自持有一組 CSR 暫存器
- 用於設定或是紀錄 CPU 的運作狀況
- 本議程只會提到 Machine Mode 下的 CSR
- 預設情況下，任何 Mode 發生的異常都會導向 M mode
  - 可以透過設置 mideleg/medeleg 將異常委託給 S mode 處理

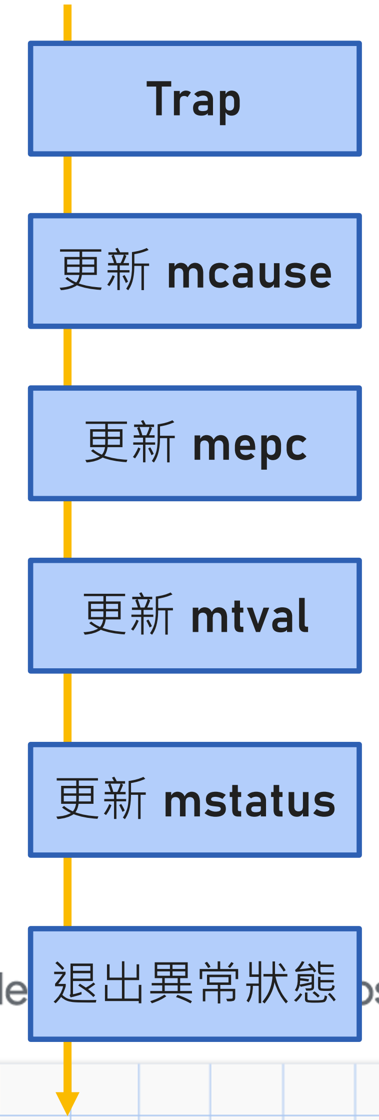
Reference: The RISC-V Instruction Set Manual

Control and Status Register

- mtvec: 當進入異常時，PC (Program counter) 會進入 mtvec 所指向的地址繼續執行。
- mcause: 紀載異常的原因。
- mtval: 紀載異常訊息。
- mepc: 進入異常前 PC 所指向的地址。若異常處理完畢，Program counter 可以讀取該位址繼續執行。
- mstatus: 進入異常時，硬體會更新 mstatus 寄存器的某些域值。
- mie: 決定中斷是否被處理。
- mip: 反映不同類型中斷的等待狀態。

Reference: The RISC-V Instruction Set Manual

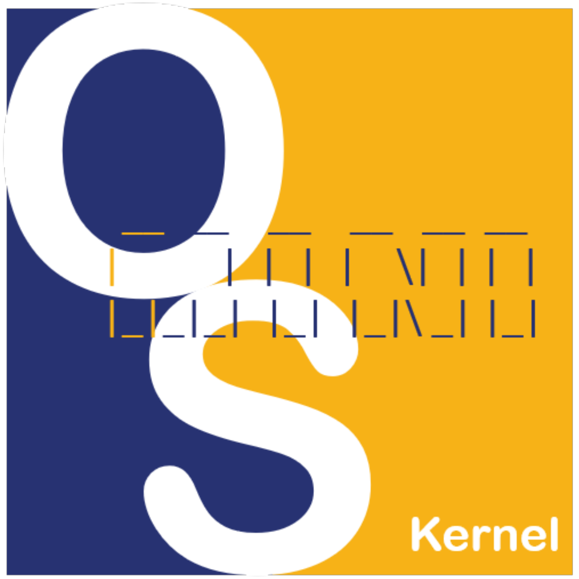
Interrupt & Exception Handle



- 作業系統會維護一份中斷向量表 (mtvec 會儲存它的位址)
- 當中斷 or 異常發生，會跳進 mtvec 所指的位址開始執行
- 硬體會記錄中斷發生的原因，並更新到 mcause
- 中斷發生前，處理器本來要執行的下一條指令 (PC + 4)
- 紀錄造成異常發生的暫存器訪問地址或是指令編碼
- 中斷 Enable (MIE)、特權模式間的切換 (MPP)
- 執行 mepc 紀錄的指令位址
- 更新 mstatus (MIE = MPIE, MPIE = 1)

Intro: mini-riscv-os

- 執行在 RISC-V 平台上的輕量 OS Kernel
- 受到 mini-arm-os 啟發
- 該專案由陳鍾誠老師發起
- 完整的程式碼範例和中文文檔
- 規模大概 1000 行 (易讀好學)

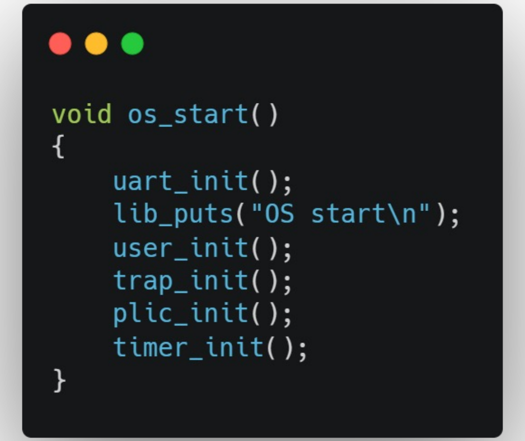


https://github.com/cccriscv/mini-riscv-os

Trace the source code !

探討目標: MINI-RISCV-OS 的中斷處理 (07-ExterInterrupt)

- Timer interrupt (任務切換)
- External interrupt (鍵盤輸入)



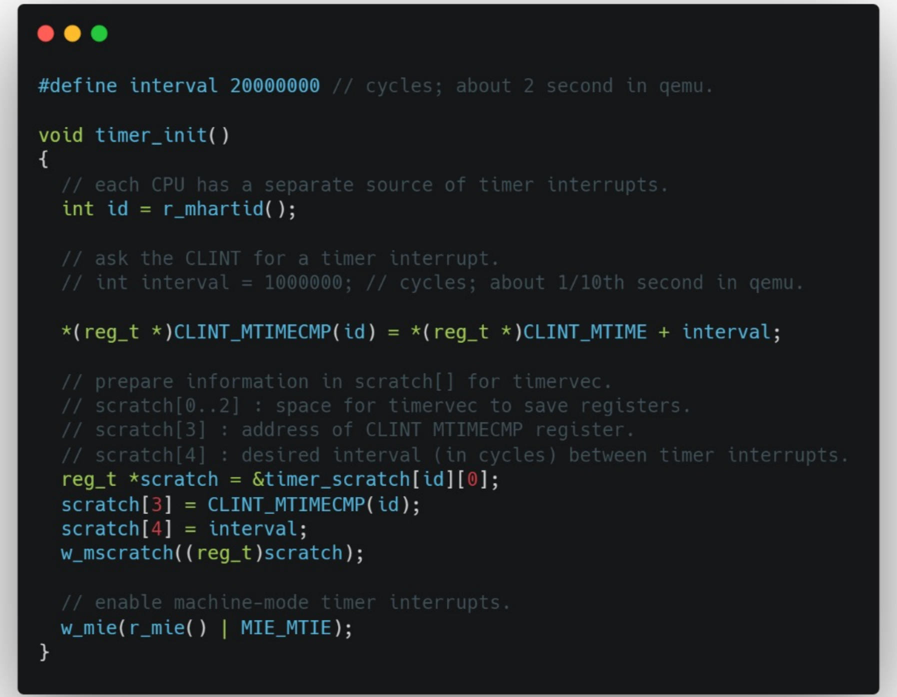
```
void os_start()
{
    uart_init();
    lib_puts("OS start\n");
    user_init();
    trap_init();
    plic_init();
    timer_init();
}
```

Google Developer Groups

https://github.com/cccriscv/mini-riscv-os/blob/master/07-ExterInterrupt/os.c

Timer interrupt

- RISC-V 有兩組暫存器用於觸發時間中斷:
  - MTIME
  - MTIMECMP
- MTIME > MTIMECMP => 時間中斷!
- 用途?

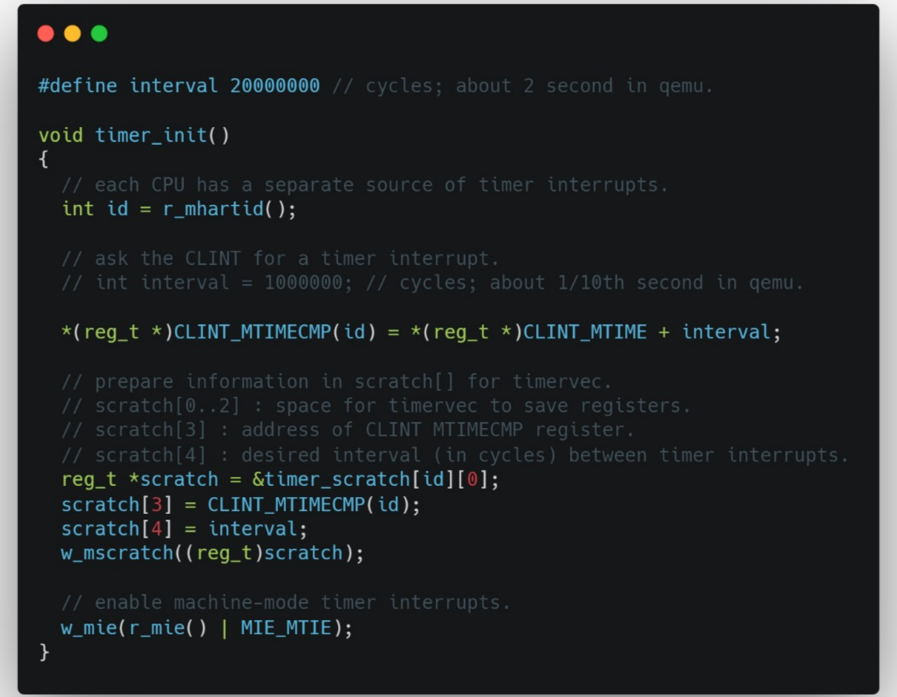


```c
#define interval 20000000 // cycles; about 2 second in qemu.

void timer_init()
{
  // 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.

  *(reg_t *)CLINT_MTIMECMP(id) = *(reg_t *)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.
  reg_t *scratch = &timer_scratch[id][0];
  scratch[3] = CLINT_MTIMECMP(id);
  scratch[4] = interval;
  w_mscratch((reg_t)scratch);

  // enable machine-mode timer interrupts.
  w_mie(r_mie() | MIE_MTIE);
}
```

Google Developer Groups

Timer interrupt

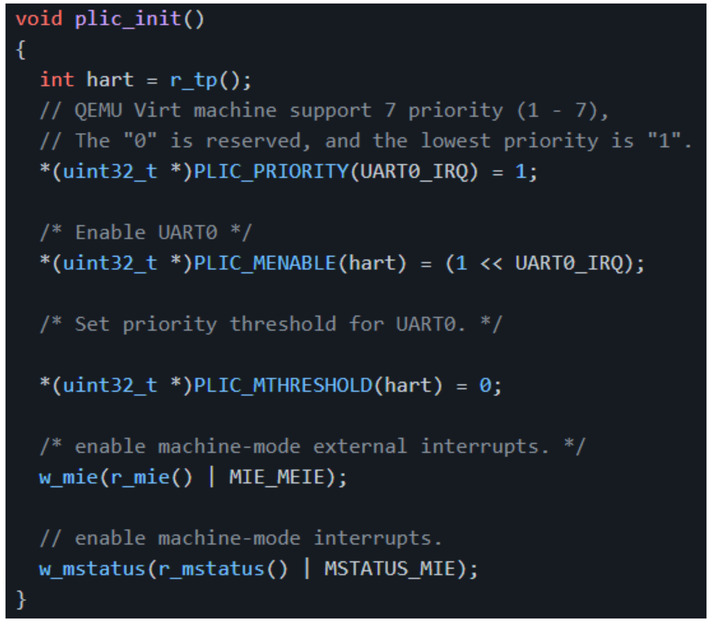
- RISC-V 有兩組暫存器用於觸發時間中斷:
  - MTIME
  - MTIMECMP
- MTIME > MTIMECMP => 時間中斷!
- 用途? 待會揭曉 m(_ _)m



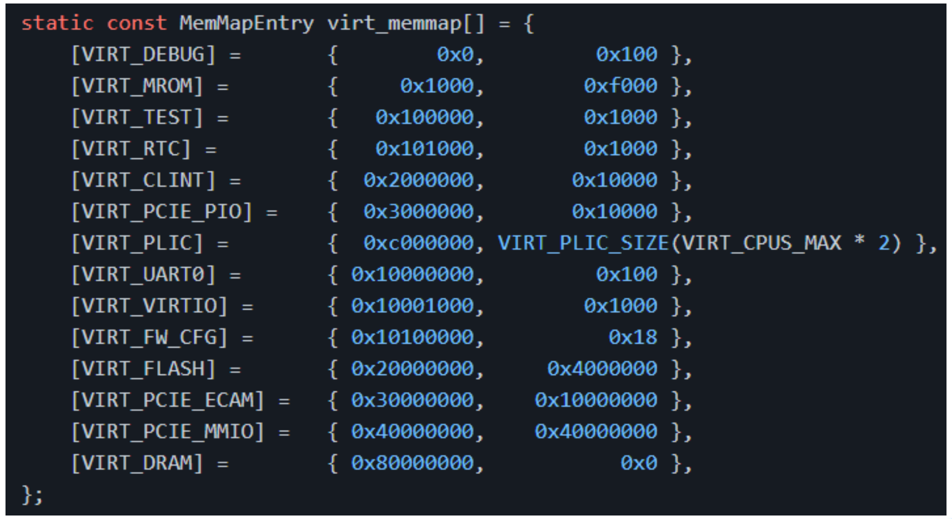
```
#define interval 20000000 // cycles; about 2 second in qemu.

void timer_init()
{
  // 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.

  *(reg_t *)CLINT_MTIMECMP(id) = *(reg_t *)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.
  reg_t *scratch = &timer_scratch[id][0];
  scratch[3] = CLINT_MTIMECMP(id);
  scratch[4] = interval;
  w_mscratch((reg_t)scratch);

  // enable machine-mode timer interrupts.
  w_mie(r_mie() | MIE_MTIE);
}
```

External Interrupt

- PLIC (Platform-Level Interrupt Controller) 是為了 RISC-V 平台所打造的 PIC。
- 設定 IRQ 的 Priority (VirtIO IRQ, UART IRQ...)
- 0xc000000 (PLIC_BASE) + offset = Mapped Address of register



```c
void plic_init()
{
  int hart = r_tp();
  // QEMU Virt machine support 7 priority (1 - 7),
  // The "0" is reserved, and the lowest priority is "1".
  *(uint32_t *)PLIC_PRIORITY(UART0_IRQ) = 1;

  /* Enable UART0 */
  *(uint32_t *)PLIC_MENABLE(hart) = (1 << UART0_IRQ);

  /* Set priority threshold for UART0. */

  *(uint32_t *)PLIC_MTHRESHOLD(hart) = 0;

  /* enable machine-mode external interrupts. */
  w_mie(r_mie() | MIE_MEIE);

  // enable machine-mode interrupts.
  w_mstatus(r_mstatus() | MSTATUS_MIE);
}
```

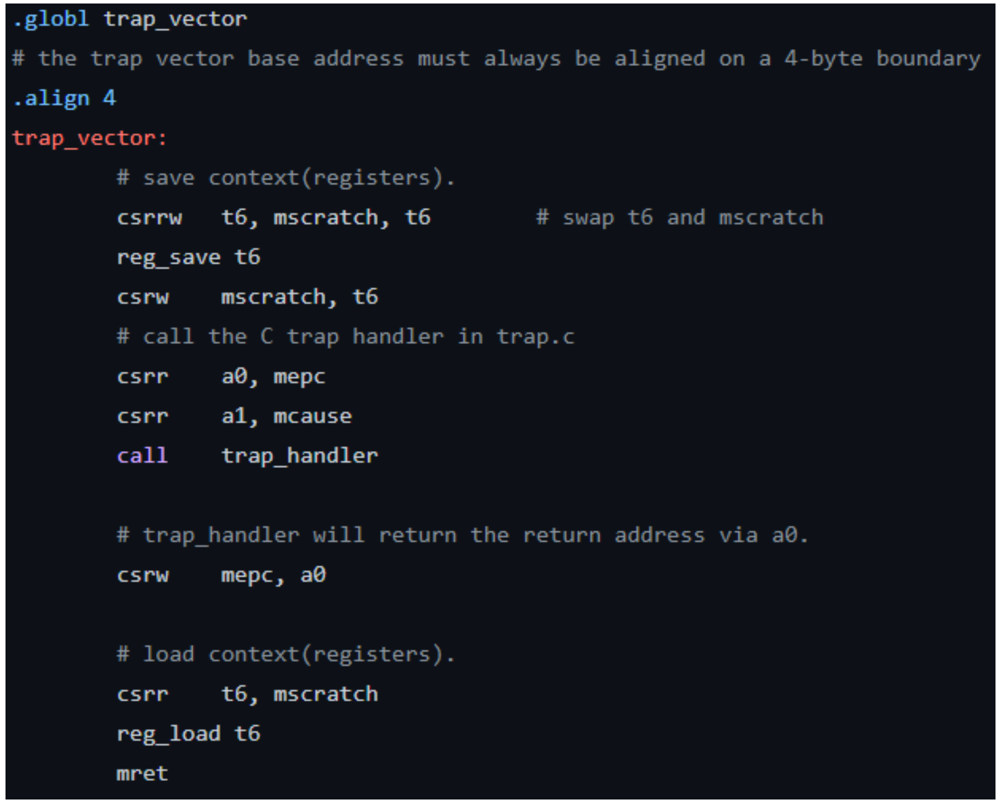


```c
static const MemMapEntry virt_memmap[] = {
    [VIRT_DEBUG] =       {        0x0,         0x100 },
    [VIRT_MROM] =        {     0x1000,        0xf000 },
    [VIRT_TEST] =        {   0x100000,        0x1000 },
    [VIRT_RTC] =         {   0x101000,        0x1000 },
    [VIRT_CLINT] =       {  0x2000000,       0x10000 },
    [VIRT_PCIE_PIO] =    {  0x3000000,       0x10000 },
    [VIRT_PLIC] =        {  0xc000000, VIRT_PLIC_SIZE(VIRT_CPUS_MAX * 2) },
    [VIRT_UART0] =       { 0x10000000,         0x100 },
    [VIRT_VIRTIO] =      { 0x10001000,        0x1000 },
    [VIRT_FW_CFG] =      { 0x10100000,          0x18 },
    [VIRT_FLASH] =       { 0x20000000,     0x4000000 },
    [VIRT_PCIE_ECAM] =   { 0x30000000,    0x10000000 },
    [VIRT_PCIE_MMIO] =   { 0x40000000,    0x40000000 },
    [VIRT_DRAM] =        { 0x80000000,           0x0 },
};
```

https://github.com/qemu/qemu/blob/master/hw/riscv/virt.c

Trap Handler

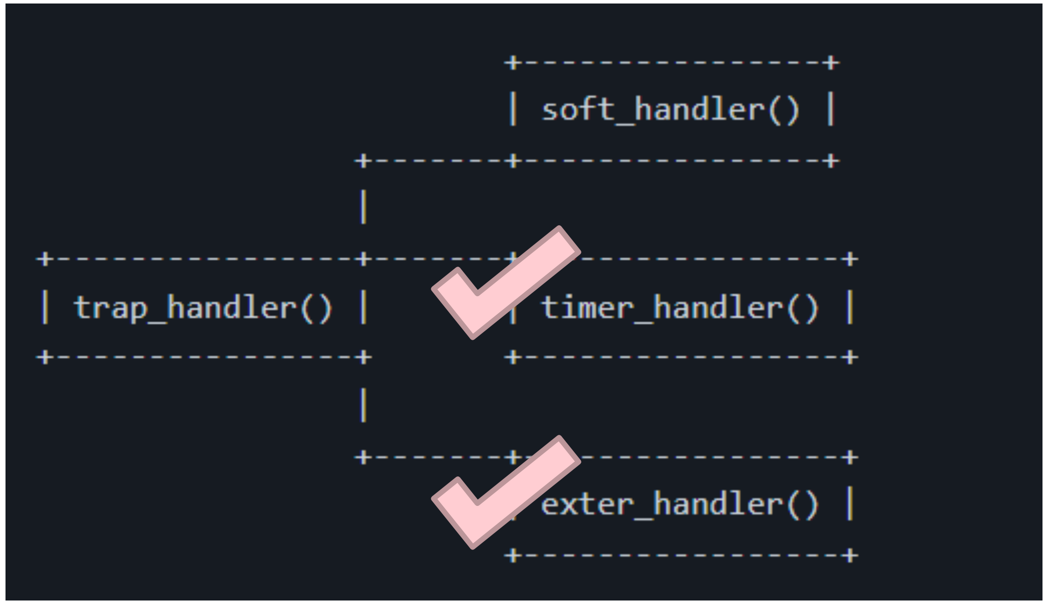
- 如何讓系統在發生中斷時跳到 trap_handler
  - trap_init
  - trap_vector
  - trap_handler
- mscratch 的用途?
  - 每個 mode 都有各自的 stack
  - 在其他 ISA，每個 mode 都有各自的 sp



```
.globl trap_vector
# the trap vector base address must always be aligned on a 4-byte boundary
.align 4
trap_vector:
        # save context(registers).
        csrrw   t6, mscratch, t6        # swap t6 and mscratch
        reg_save t6
        csrw    mscratch, t6
        # call the C trap handler in trap.c
        csrr    a0, mepc
        csrr    a1, mcause
        call    trap_handler

        # trap_handler will return the return address via a0.
        csrw    mepc, a0

        # load context(registers).
        csrr    t6, mscratch
        reg_load t6
        mret
```

Trap Handler (Cont'd)

- 有點像是 Dispatcher
- 根據 mcause 判斷中斷或是異常的類型
  - Timer interrupt
  - External interrupt

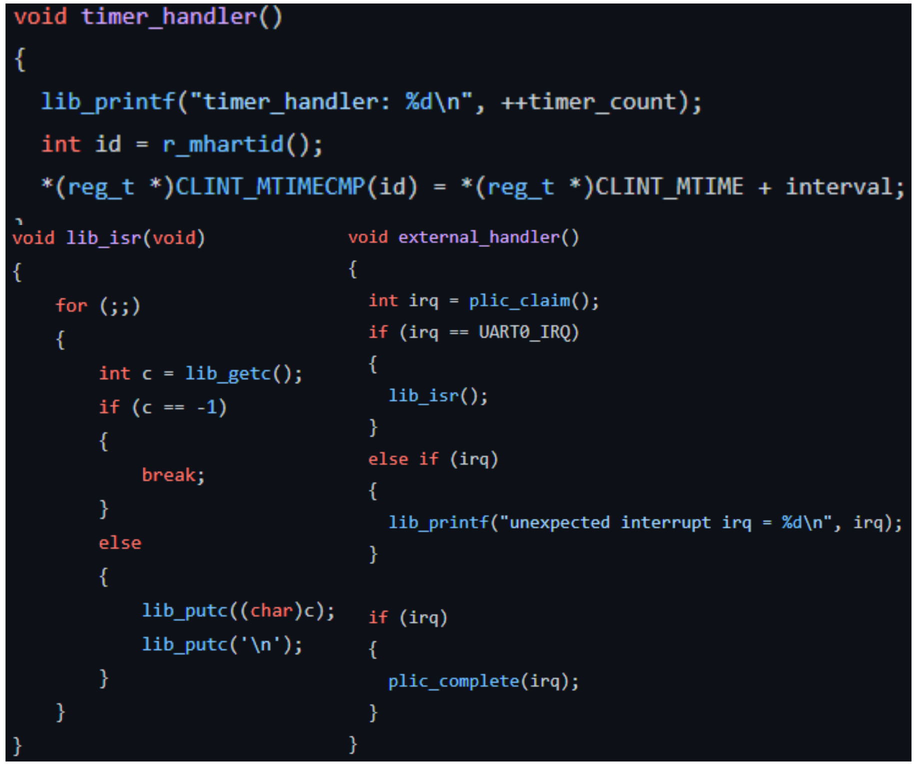


```
                               +----------------+
                               | soft_handler() |
                      +--------+----------------+
                      |
+----------------+----+--------+----------------+
| trap_handler() |             | timer_handler() |
+----------------+             +----------------+
                      |
                      +--------+----------------+
                                exter_handler() |
                               +----------------+
```

Google Developer Groups

https://github.com/cccriscv/mini-riscv-os/blob/master/07-ExterInterrupt/trap.c

Trap Handler (Cont’d)



```c
void timer_handler()
{
  lib_printf("timer_handler: %d\n", ++timer_count);
  int id = r_mhartid();
  *(reg_t *)CLINT_MTIMECMP(id) = *(reg_t *)CLINT_MTIME + interval;
```

```c
void lib_isr(void)
{
    for (;;)
    {
        int c = lib_getc();
        if (c == -1)
        {
            break;
        }
        else
        {
            lib_putc((char)c);
            lib_putc('\n');
        }
    }
}
```

```c
void external_handler()
{
  int irq = plic_claim();
  if (irq == UART0_IRQ)
  {
    lib_isr();
  }
  else if (irq)
  {
    lib_printf("unexpected interrupt irq = %d\n", irq);
  }

  if (irq)
  {
    plic_complete(irq);
  }
}
```

- external_handler 會判斷 IRQ 再指派 ISR。
  - 擴充性
- timer_handler 加大 MTIMECMP 的 Value。

Google Developer Groups

https://github.com/cccriscv/mini-riscv-os/blob/master/07-ExterInterrupt/timer.c

Trap Handler (Cont’d)

- Timer interrupt 的用途?
  A: 我是無情的原始碼閱讀機器。

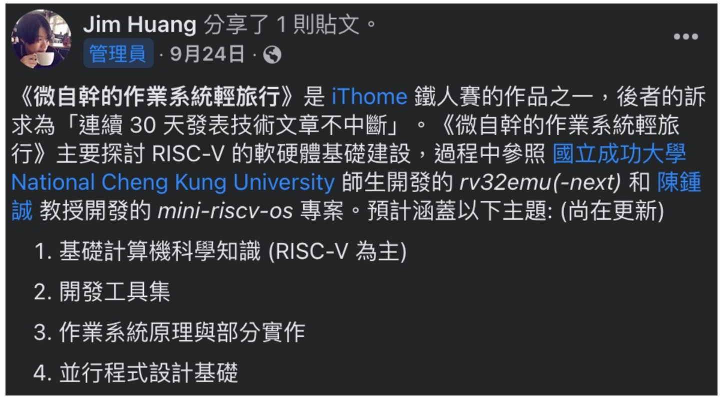
https://github.com/cccriscv/mini-riscv-os/blob/master/07-ExterInterrupt/trap.c

Why we aren’t to make a Shell

- Interruption Handler
- Disk Driver
- File system
- System call (fork, wait, exec...)
- Process 的實作不夠完整
- Shell

Acquisition

- 從 RISC-V 處理器到 UNIX 作業系統
- 文章能被看見
- “微”懂作業系統
- 建立基礎後可以繼續挑戰 Emulator, Compiler... 的實作



Jim Huang 分享了 1 則貼文。
管理員 · 9月24日 ·

《微自幹的作業系統輕旅行》是 iThome 鐵人賽的作品之一，後者的訴求為「連續 30 天發表技術文章不中斷」。《微自幹的作業系統輕旅行》主要探討 RISC-V 的軟硬體基礎建設，過程中參照 國立成功大學 National Cheng Kung University 師生開發的 *rv32emu(-next)* 和 陳鍾誠 教授開發的 *mini-riscv-os* 專案。預計涵蓋以下主題: (尚在更新)

1. 基礎計算機科學知識 (RISC-V 為主)
2. 開發工具集
3. 作業系統原理與部分實作
4. 並行程式設計基礎

Google Developer Groups

Thanks for your listening !

Any further question?