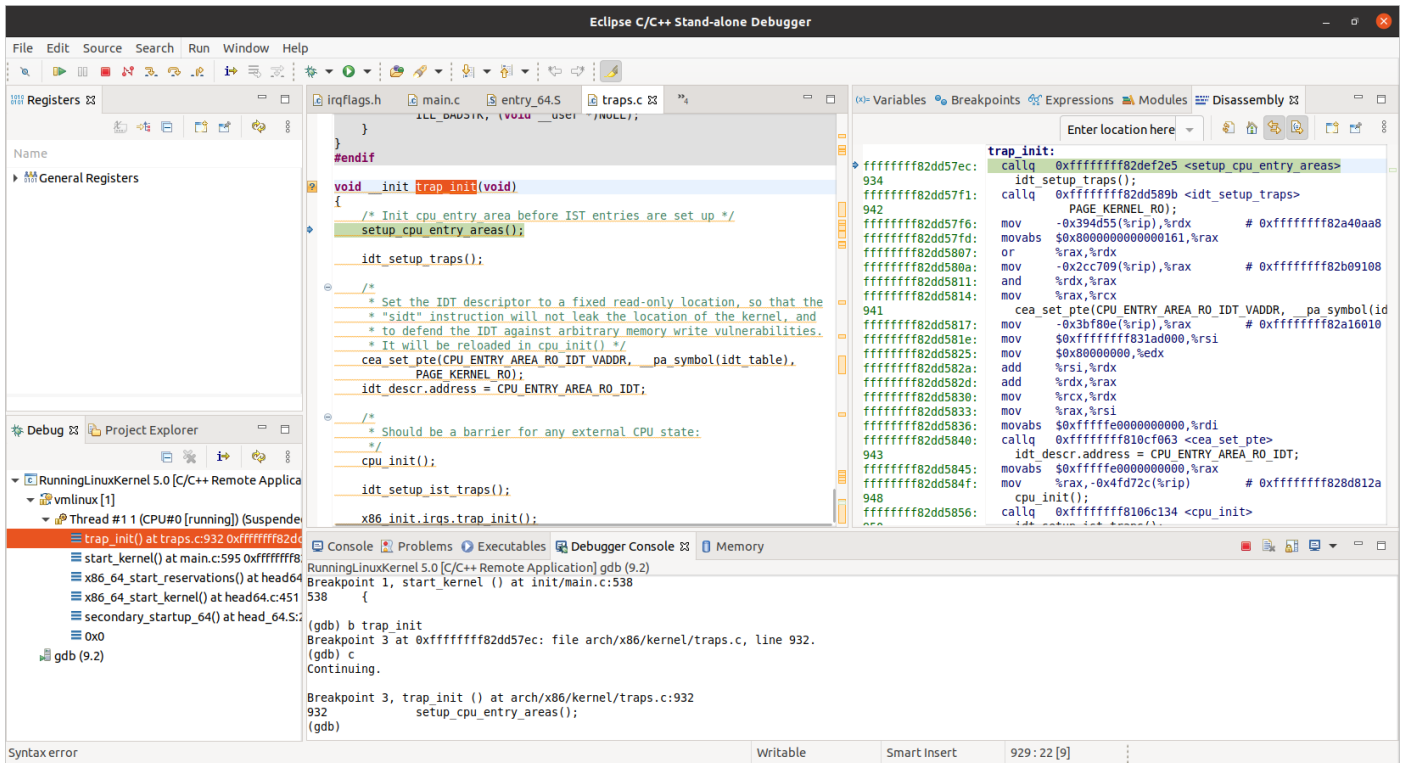


作業系統概論 hw1

學號: 408410113 姓名: 王 X 彥

1. b trap_init



```
void __init trap_init(void)
{
    /* Init cpu entry area before IST entries are set up */
    setup_cpu_entry_areas();

    idt_setup_traps();

    /*
     * Set the IDT descriptor to a fixed read-only location, so that the
     * "sidt" instruction will not leak the location of the kernel, and
     * to defend the IDT against arbitrary memory write vulnerabilities.
     * It will be reloaded in cpu_init().
     */
    cea_set_pte(CPU_ENTRY_AREA_RO_IDT_VADDR, __pa_symbol(idt_table),
                PAGE_KERNEL_RO);
    idt_descr.address = CPU_ENTRY_AREA_RO_IDT;

    /*
     * Should be a barrier for any external CPU state:
     */
    cpu_init();

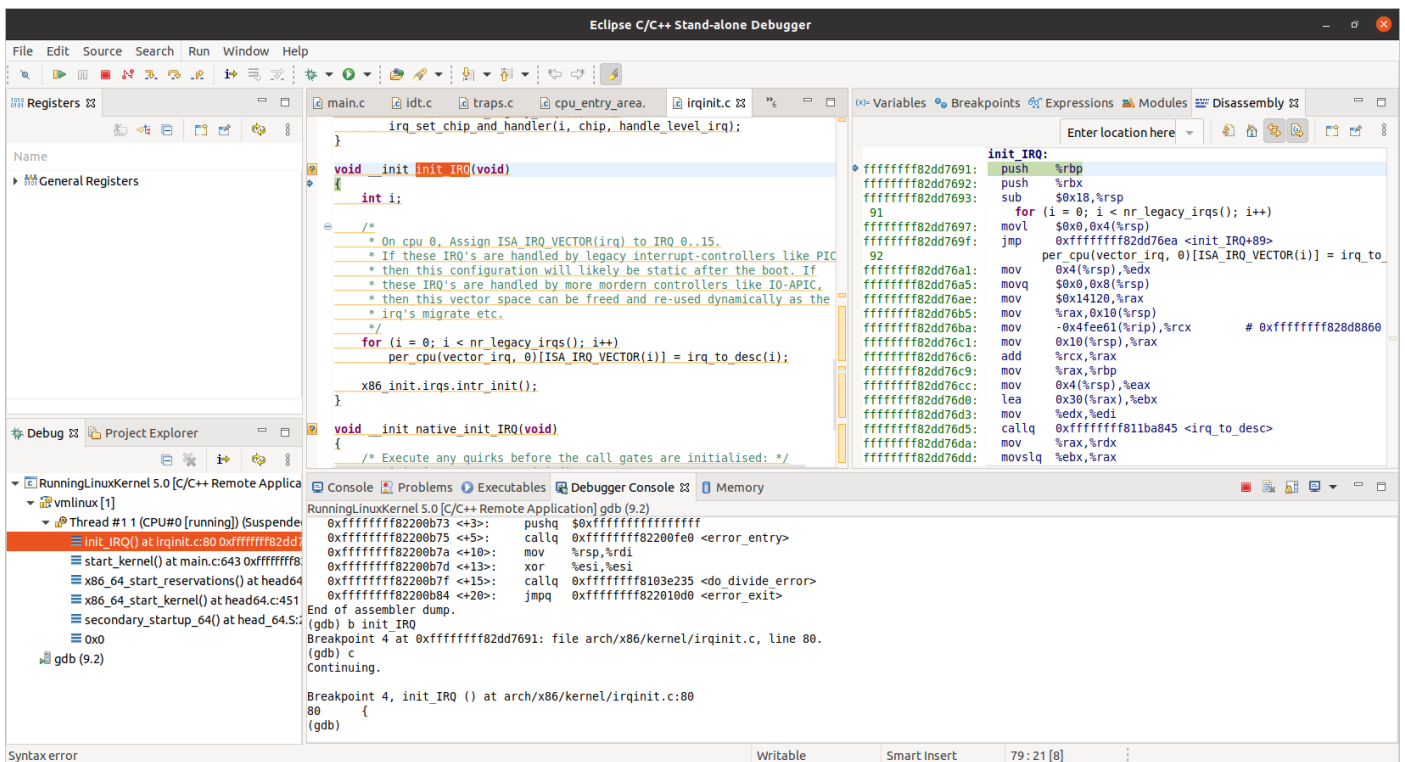
    idt_setup_ist_traps();

    x86_init.irqs.trap_init();
}
```

Disassembly of trap_init:

```
trap_init:
  callq 0xffffffff82def2e5 <setup_cpu_entry_areas>
  idt_setup_traps();
  callq 0xffffffff82dd589b <idt_setup_traps>
  mov     -0x394d55(%rip),%rdx    # 0xffffffff82a40aa8
  movabs  $0x8000000000000161,%rax
  or      %rax,%rdx
  mov     -0x2cc709(%rip),%rax    # 0xffffffff82b09108
  and     %rdx,%rax
  mov     %rax,%rcx
  cea_set_pte(CPU_ENTRY_AREA_RO_IDT_VADDR, __pa_symbol(idt_table),
              PAGE_KERNEL_RO);
  mov     -0x3bf80e(%rip),%rax    # 0xffffffff82a16010
  mov     $0xffffffff831ad000,%rsi
  add     %rsi,%rdx
  add     %rdx,%rax
  mov     %rcx,%rdx
  mov     %rax,%rsi
  movabs  $0xffffffff80000000,%rdi
  callq 0xffffffff810cf63 <cea_set_pte>
  idt_descr.address = CPU_ENTRY_AREA_RO_IDT;
  movabs  $0xffffffff80000000,%rax
  mov     %rax,-0x47d2c(%rip)     # 0xffffffff828d812a
  callq 0xffffffff8106c134 <cpu_init>
  idt_setup_ist_traps();
```

2. b init_IRQ



```
void __init init_IRQ(void)
{
    int i;

    /*
     * On cpu 0, Assign ISA_IRQ_VECTOR(irq) to IRQ 0..15.
     * If these IRQ's are handled by legacy interrupt-controllers like PIC
     * then this configuration will likely be static after the boot. If
     * these IRQ's are handled by more modern controllers like IO-APIC,
     * then this vector space can be freed and re-used dynamically as the
     * irq's migrate etc.
     */
    for (i = 0; i < nr_legacy_irqs(); i++)
        per_cpu(vector_irq, 0)[ISA_IRQ_VECTOR(i)] = irq_to_desc(i);

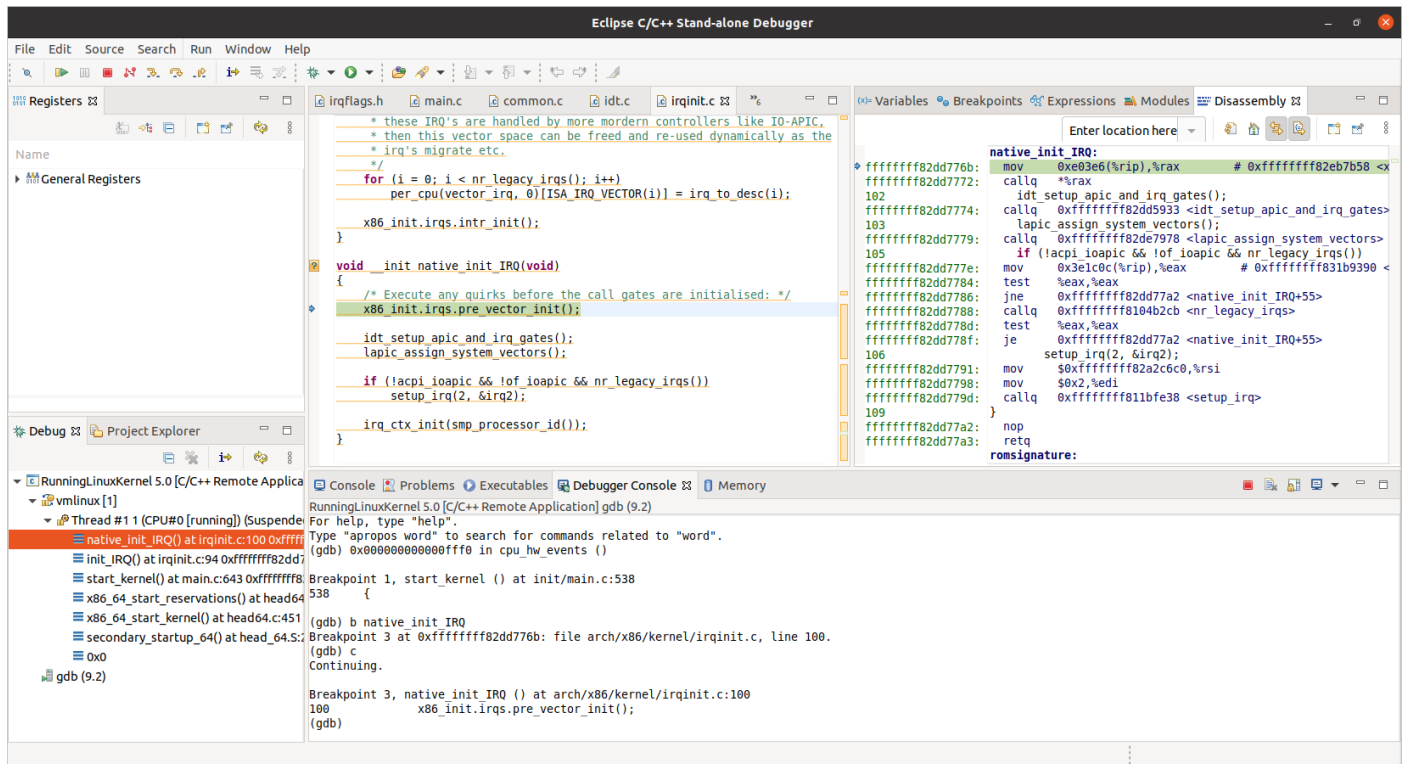
    x86_init.irqs.intr_init();
}

void __init native_init_IRQ(void)
{
    /* Execute any quirks before the call gates are initialised: */
}
```

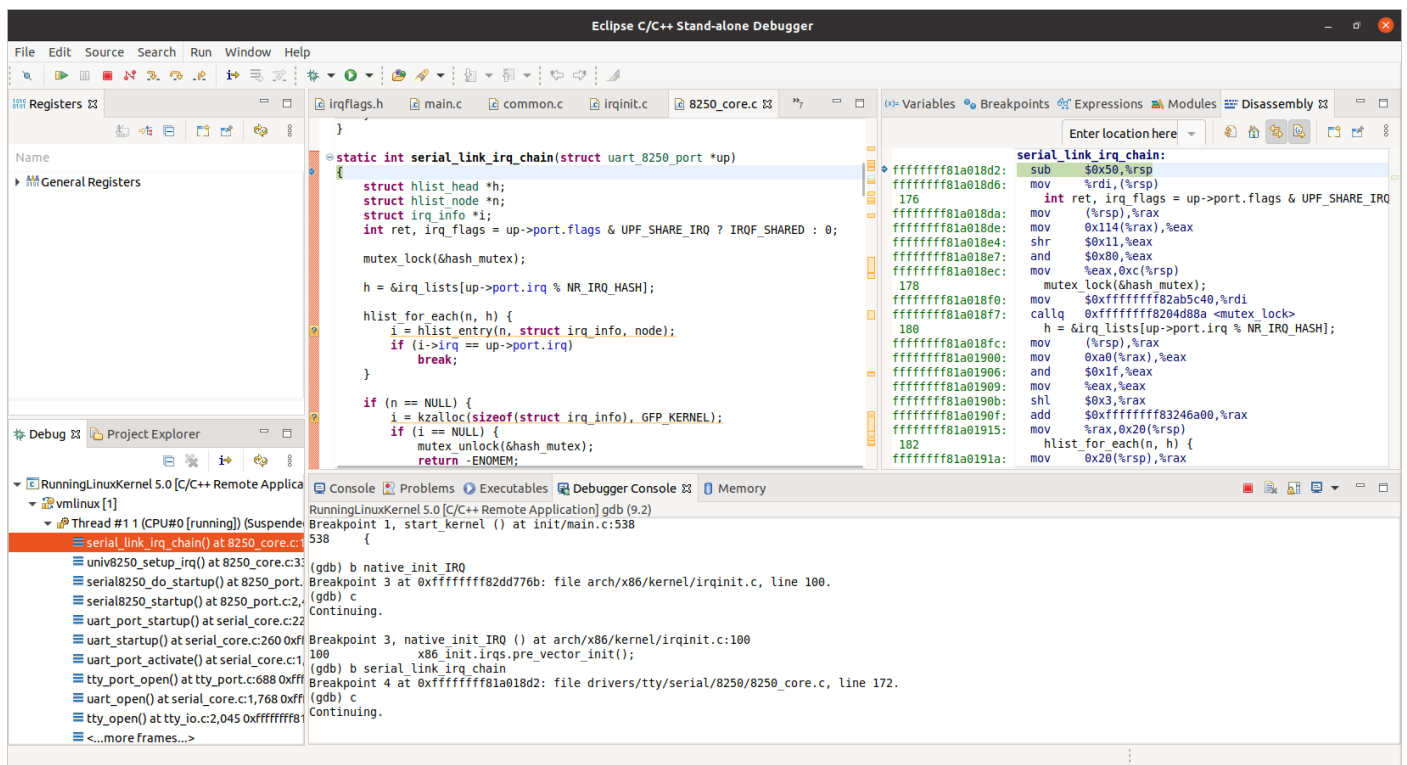
Disassembly of init_IRQ:

```
init_IRQ:
  push    %rbp
  push    %rbx
  sub     $0x18,%rsp
  movl    $0x0,0x4(%rsp)
  jmp     0xffffffff82dd7697 <init_IRQ+89>
  per_cpu(vector_irq, 0)[ISA_IRQ_VECTOR(i)] = irq_to_desc(i);
  mov     0x4(%rsp),%edx
  movq    $0x0,0x8(%rsp)
  mov     $0x14120,%rax
  mov     %rax,0x10(%rsp)
  mov     -0x4fee61(%rip),%rcx    # 0xffffffff828d8860
  mov     0x10(%rsp),%rax
  add     %rcx,%rax
  mov     %rax,%rbp
  mov     0x4(%rsp),%eax
  lea     0x30(%rax),%ebx
  mov     %edx,%edi
  callq 0xffffffff811ba845 <irq_to_desc>
  mov     %rax,%dx
  movslq  %ebx,%rax
```

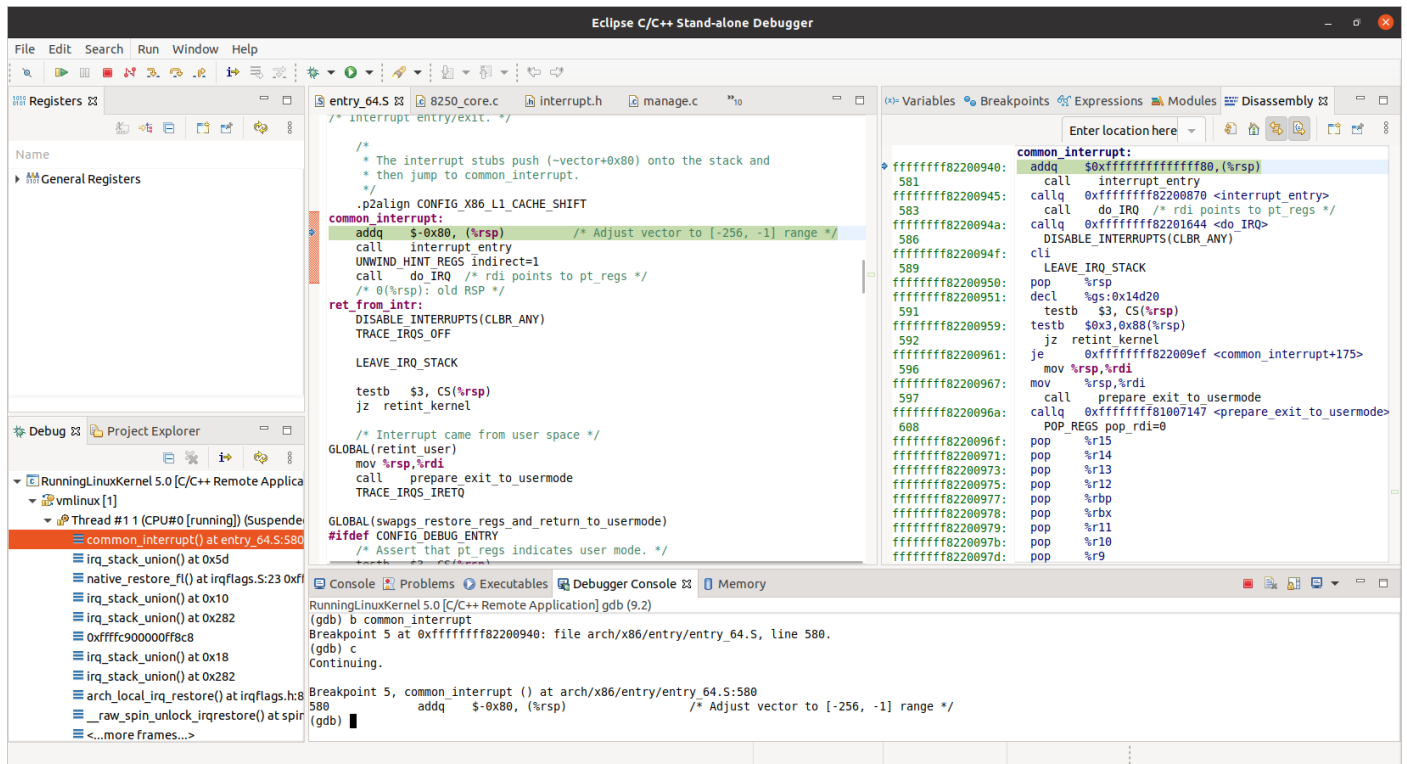
3. b native_init_IRQ



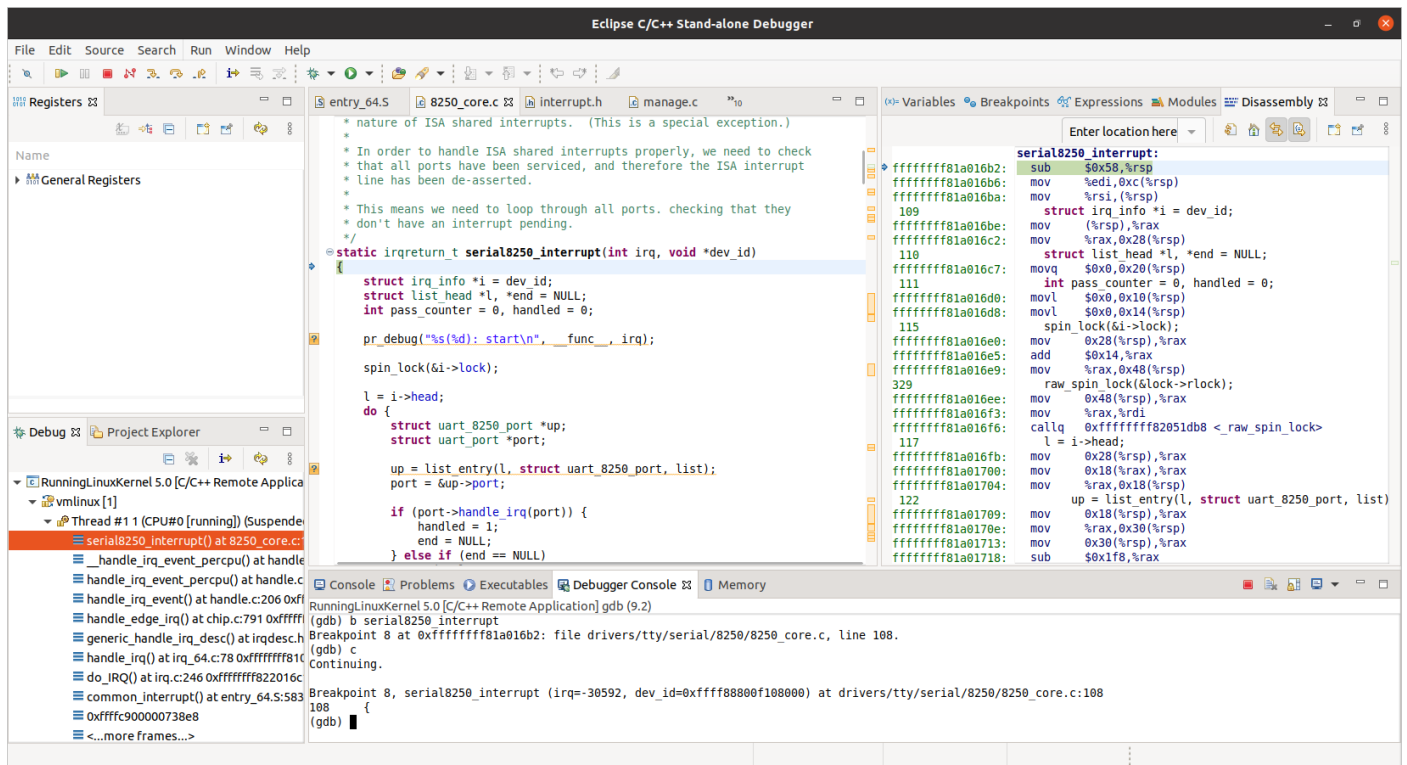
4. b serial_link_irq_chain



5. b common_interrupt



6. b serial8250_interrupt



7. disass irq_entries_start

The screenshot shows the Eclipse C/C++ Stand-alone Debugger interface. The main window displays the disassembly of the `irq_entries_start` function. The assembly code is as follows:

```
irq_entries_start:
ffffff82200210: pushq   $0x5f
ffffff82200212: jmpq    0xffffffff82200940 <common_interrupt>
ffffff82200217: nop
ffffff82200218: pushq   $0x5e
ffffff8220021a: jmpq    0xffffffff82200940 <common_interrupt>
ffffff8220021f: nop
ffffff82200220: pushq   $0x5d
ffffff82200222: jmpq    0xffffffff82200940 <common_interrupt>
ffffff82200227: nop
ffffff82200228: pushq   $0x5c
ffffff8220022a: jmpq    0xffffffff82200940 <common_interrupt>
ffffff8220022f: nop
ffffff82200230: pushq   $0x5b
ffffff82200232: jmpq    0xffffffff82200940 <common_interrupt>
ffffff82200237: nop
ffffff82200238: pushq   $0x5a
ffffff8220023a: jmpq    0xffffffff82200940 <common_interrupt>
ffffff8220023f: nop
ffffff82200240: pushq   $0x59
ffffff82200242: jmpq    0xffffffff82200940 <common_interrupt>
ffffff82200247: nop
ffffff82200248: pushq   $0x58
ffffff8220024a: jmpq    0xffffffff82200940 <common_interrupt>
ffffff8220024f: nop
ffffff82200250: pushq   $0x57
ffffff82200252: jmpq    0xffffffff82200940 <common_interrupt>
ffffff82200257: nop
ffffff82200258: pushq   $0x56
ffffff8220025a: jmpq    0xffffffff82200940 <common_interrupt>
ffffff8220025f: nop
ffffff82200260: pushq   $0x55
ffffff82200262: jmpq    0xffffffff82200940 <common_interrupt>
ffffff82200267: nop
```

The left sidebar shows the Project Explorer with the `entry_64.S` file selected. The bottom console shows the output of the debugger, indicating that the assembly dump is complete.

8. b *(irq_entries_start+56)

The screenshot shows the Eclipse C/C++ Stand-alone Debugger interface. The main window displays the disassembly of the `irq_entries_start` function. The assembly code is as follows:

```
ENTRY(irq_entries_start)
vector=FIRST_EXTERNAL_VECTOR
.rept (FIRST_SYSTEM_VECTOR - FIRST_EXTERNAL_VECTOR)
UNWIND_HINT_IRET_REGS
pushq   $(-vector+0x80) /* Note: always in signed byte range */
jmp common_interrupt
.align 8
vector=vector+1
.endr
END(irq_entries_start)

.macro DEBUG_ENTRY_ASSERT_IRQS_OFF
#ifdef CONFIG_DEBUG_ENTRY
pushq   %rax
SAVE_FLAGS(CLBR_RAX)
testl   $X86_EFLAGS_IF, %eax
jz .Lokay_@
ud2
.Lokay_@:
popq    %rax
#endif
.endm
/*
```

The left sidebar shows the Project Explorer with the `entry_64.S` file selected. The bottom console shows the output of the debugger, indicating that the assembly dump is complete. A breakpoint is set at the instruction `b *(irq_entries_start+56)`, which is highlighted in the disassembly window.

問題2. 說明Linux如何設定中斷向量

CPU內建的中斷。CPU內建的「中斷事件」，也稱作「software interrupt」或「trap」。Linux在start_kernel中，先呼叫trap_init，將CPU內部中斷的中斷處理函數寫入到「中斷向量表」（interrupt vector table），共19個

外部的中斷。先呼叫init_IRQ之後

native_init_IRQ->idt_setup_apic_and_irq_gates->idt_setup_from_table->idt_init_desc初始化傳統的16個外部中斷。

問題3. 說明Linux如何從中斷向量的組合語言部分（interrupt service routine，這裡只討論外部中斷）跳躍到特定的中斷函數

經過一連串的trace可以得知中斷的編號放在orig_ax，因此可以當作vector_irq的index，並透過__this_cpu_read(vector_irq[vector])獲得結構irq_desc，從結構裡desc->action->handler(...)可以獲得期中斷函數。