PA3 实验报告（完成至 3.1）

221300066

2023 年 11 月 30 日

1 穿越时空的旅程

触发自陷操作

实现异常响应机制 实现新指令（inst.c 中）：

```
INSTPAT("??????? ????? ????? 010 ????? 11100 11", csrrs , I, csrrs_instrcti
INSTPAT("??????? ????? ????? 001 ????? 11100 11", csrrw , I, csrrw_instrcti
INSTPAT("0011000 00010 00000 000 00000 11100 11", mret , R, s->dnpc = (vad
INSTPAT("0000000 00000 00000 000 00000 11100 11", ecall , I, s->dnpc = isa_
```

实现 isa_raise_intr() 函数：

```
word_t isa_raise_intr(word_t NO, vaddr_t epc)
{
  cpu.csr[REG_MEPC]._32 = epc;
  cpu.csr[REG_MCAUSE]._32 = NO;
  word_t mtvec = cpu.csr[REG_MTVEC]._32;
  word_t mode = mtvec & 0x1;
  word_t base = mtvec & (~0x3);
  word_t offset = NO * 4;
  word_t vec = (mode == 0) ? base : (base + offset);
  // cpu.pc = vec;
  if (cpu.csr[REG_MSTATUS]._32 & MSTATUS_MIE)
    cpu.csr[REG_MSTATUS]._32 |= MSTATUS_MPIE;
  else
```

```
    cpu.csr[REG_MSTATUS]._32 &= (~MSTATUS_MPIE);
  cpu.csr[REG_MSTATUS]._32 &= (~MSTATUS_MIE);
  return vec;
}
```

保存上下文

重新组织 Context 结构体 参考 abstract-machine/am/src/riscv32/nemu/trap.S 中构造的上下文易得:

```
struct Context {
  // TODO: fix the order of these members to match trap.S
  uintptr_t gpr[NR_REGS], mcause, mstatus , mepc;
  void *pdir;
};
```

必答题（需要在实验报告中回答）- 理解上下文结构体的前世今生
___am_irq_handle() 是一个中断处理函数，它的参数 c 是一个上下文结构指针，用来保存和恢复中断前后的寄存器状态。c 指向的上下文结构是在栈上分配的。具体地，是在 trap.S 中的 ___alltraps 函数中，使用 addi 指令将 sp 寄存器减去一个偏移量（CONTEXT_SIZE），从而为上下文结构预留空间。然后，使用 sd 指令将各个寄存器的值保存到 sp 寄存器指向的地址上，形成上下文结构。最后，使用 mv 指令将 sp 寄存器的值传递给 a0 寄存器，作为 ___am_irq_handle() 的参数 c。
上下文结构有很多成员，分别对应不同的寄存器。每一个成员都是在 trap.S 中的 ___alltraps 函数中赋值的，具体地，是使用 sd 指令将寄存器的值保存到 sp 寄存器指向的地址上，然后将 sp 寄存器加上一个偏移量（8 字节），指向下一个成员的地址。例如，以下是保存 x1 寄存器（ra）和 x2 寄存器（sp）的代码:

```
# save x1
sd x1, 0(sp)
addi sp, sp, 8
# save x2
```

```
  sd x2, 0(sp)
  addi sp, sp, 8
```

ISA-nemu.h, trap.S, 上述讲义文字, 以及你刚刚在 NEMU 中实现的新指令, 这四部分内容都是关于中断处理的。ISA-nemu.h 是一个头文件, 定义了上下文结构的类型和大小, 以及一些中断相关的宏和函数。trap.S 是一个汇编文件, 实现了中断向量表和中断入口函数, 以及保存和恢复上下文结构的代码。上述讲义文字是对中断处理的原理和流程的解释, 以及一些练习题和提示。你刚刚在 NEMU 中实现的新指令是 mret 指令, 它是用来从中断处理程序返回到中断前的程序的, 它的功能是将 MEPC 寄存器的值恢复到 PC 寄存器, 以及将 MSTATUS 寄存器的 MPIE 位恢复到 MIE 位。

事件分发

识别自陷事件 ___am_irq_handle() 的代码会把执行流切换的原因打包成事件, 然后调用在 cte_init() 中注册的事件处理回调函数, 将事件交给 Nanos-lite 来处理:

```
Context* ___am_irq_handle(Context *c) {
  if (user_handler) {
    Event ev = {0};
    switch (c->mcause) {
      default: ev.event = EVENT_ERROR; break;
    }
    c = user_handler(ev, c);
    assert(c != NULL);
  }
  return c;
}
```

恢复上下文

恢复上下文 恢复上下文最重要的是恢复 pc, 触发中断的指令地址保存在 mepc 中, 那么 mret 就负责从用这个寄存器恢复 pc。

```
word_t isa_raise_intr(word_t NO, vaddr_t epc) {
  /* TODO: Trigger an interrupt/exception with ``NO''.
   * Then return the address of the interrupt/exception vector.
  if(NO==0){
    epc+=4;
  }
  cpu.csr.mcause = NO;
  cpu.csr.mepc = epc;
  return cpu.csr.mtvec;
}
```

必答题（需要在实验报告中回答）- 理解穿越时空的旅程 yield test 调用 yield() 开始，到从 yield() 返回的期间，这一趟旅程具体经历了以下几个步骤：

1. yield test 程序在用户态执行 yield() 函数，该函数的定义在 am/src/nemu-common/nemu-trap.c 文件中，它的功能是触发一个系统调用（ecall 指令），并将系统调用号（SYS_yield）和参数（NULL）传递给内核态。
2. NEMU 在执行 ecall 指令时，会检测到当前的特权模式是用户态（U-mode），并且当前的中断使能位（MIE）是开启的，因此会触发一个用户态软件中断（U-mode software interrupt），并将中断号（0x8）和中断前的程序计数器（PC）保存到 MCAUSE 和 MEPC 寄存器中，然后跳转到中断向量地址执行中断处理程序。
3. 中断向量地址是由 MTVEC 寄存器指定的，它的值是在 am/src/nemu-common/nemu-init.c 文件中的 _init() 函数中设置的，它指向了 am/src/nemu-common/trap.S 文件中的 ___alltraps 函数，该函数是一个通用的中断入口函数，它的功能是保存和恢复中断前后的寄存器状态，以及调用具体的中断处理函数。
4. ___alltraps 函数首先会使用 addi 指令将 sp 寄存器减去一个偏移量（CONTEXT_SIZE），从而为上下文结构预留空间。然后，使用 sd 指令将各个寄存器的值保存到 sp 寄存器指向的地址上，形成上下文结构。最后，使用 mv 指令将 sp 寄存器的值传递给 a0 寄存器，作为 ___am_irq_handle() 函数的参数 c，该函数是在 am/src/nemu-common/nemu-irq.c 文件中定义的，它的功能是根据中断号（MCAUSE 寄存器的值）来分发不同的中断处

理函数。
5. ___am_irq_handle() 函数会使用 switch-case 语句来匹配中断号，对于用户态软件中断（0x8），它会调用 handle_syscall() 函数，该函数是在 am/src/nemu-common/nemu-syscall.c 文件中定义的，它的功能是根据系统调用号（a7 寄存器的值）来执行不同的系统调用函数。
6. handle_syscall() 函数会使用 switch-case 语句来匹配系统调用号，对于 SYS_yield（1），它会调用 yield() 函数，该函数是在 am/src/nemu-common/nemu-sched.c 文件中定义的，它的功能是切换当前运行的线程，即从就绪队列中选择下一个线程，并将其上下文结构的地址赋值给 current 变量，然后返回该地址。
7. ___am_irq_handle() 函数会将 yield() 函数的返回值（下一个线程的上下文结构的地址）传递给 ___alltraps 函数的返回值，然后返回到 ___alltraps 函数中。___alltraps 函数会使用 ld 指令将上下文结构中保存的寄存器的值恢复到各个寄存器中，然后使用 mret 指令从中断处理程序返回到中断前的程序，即下一个线程的程序。
8. mret 指令会将 MEPC 寄存器的值恢复到 PC 寄存器，以及将 MSTATUS 寄存器的 MPIE 位恢复到 MIE 位，从而实现从内核态（M-mode）返回到用户态（U-mode）的功能。此时，yield test 程序从 yield() 函数返回，继续执行下一条指令。

异常处理的踪迹 - etrace

实现 etrace PA3 一阶段到此结束