Interrupt and Exception
High-level roadmap

- [basic CPU instructions] user-level threading
- [+ timer interrupt] timeshare threading
- [+ ecall exception] system call
- [+ privilege levels] memory protection
- [+ I/O bus control] disk driver, cache and file systems
P2: interrupt and exception

- [basic CPU instructions] user-level threading
- [+ timer interrupt] timeshare threading
- [+ ecall exception] system call
- [+ privilege levels] memory protection
- [+ I/O bus control] disk driver, cache and file systems
Set a timer
First glance of timer interrupt

void handler() {
    earth->tty_info("Got timer interrupt.");
    // start another timer
}

int main() {
    // register handler() as interrupt handler
    // enable timer interrupt
    // start a timer

    while(1);
}
Execution of this program

```c
void handler() {
    earth->tty_info("...");
    // start another timer
}

int main() {
    // register handler()
    // enable timer interrupt
    // start a timer

    while(1);
}
```

- **When timer is running**
  - `while(1);`

- **When timer runs out**
  - `handler()`
How to register `handler()` as interrupt handler?

- How to `start` a timer?
- How to `enable` timer interrupt?
CSR: control and status registers

- There are many registers other than x0 .. x31.
  - *machine ISA*: 32-bit or 64bit?
  - *hart ID*: the ID number of a core in a multi-core CPU
  - *interrupt control*: timer, I/O device …
# The `mtvec` CSR

The `mtvec` CSR (Mode Register) controls how exceptions are handled. It consists of two fields:

- **BASE**: A 32-bit register that specifies the address of the exception vector table.
- **MODE**: A 3-bit field that determines how exceptions are handled.

### Table 3.5: Encoding of `mtvec` MODE field.

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Direct</td>
<td>All exceptions set pc to BASE.</td>
</tr>
<tr>
<td>1</td>
<td>Vectored</td>
<td>Asynchronous interrupts set pc to BASE+4×cause.</td>
</tr>
<tr>
<td>≥2</td>
<td>—</td>
<td><strong>Reserved</strong></td>
</tr>
</tbody>
</table>

Page 26 of RISC-V manual, volume2, v1.10

Register an interrupt handler

0800280c <handler>:
  ...

08002914 <main>:
  ...
  lui    a5,0x8003  # now a5 == 0x08003000
  addi   a5,a5,-2036 # now a5 == 0x0800280c
  # csrw: control and status register write
  csrw   mtvec,a5   # now mtvec == 0x0800280c
  ...

void handler() {
    . . .
}

int main() {
    /* Register handler with direct mode */
    asm("csrw mtvec, %0" :: "r"(handler));
    . . .
}

Register an interrupt handler in C
• How to register handler() as interrupt handler?

⇒ How to start a timer?

• How to enable timer interrupt?
Core-local Interrupt (CLINT)

E31
A RISC-V core

Page 38 of Sifive FE310 manual, v19p04
mtime and mtimecmp

A timer interrupt is triggered when mtime increases to a degree that mtime == mtimecmp

CLINT
- Machine time (mtime)
- Machine time compare (mtimecmp)

E31
A RISC-V core

Page 38 of Sifive FE310 manual, v19p04
int quantum = 500000;

void handler() {
    ... Read current time
    mtimecmp_set(mtime_get() + quantum);
}

int main() {
    ... Set timer
    mtimecmp_set(mtime_get() + quantum);
    ...
}
• How to register handler() as interrupt handler?
• How to start a timer?

How to enable timer interrupt?
The **mstatus** CSR

<table>
<thead>
<tr>
<th>SD</th>
<th>WPRI</th>
<th>CSR</th>
<th>TVM</th>
<th>MXR</th>
<th>SUM</th>
<th>MPRV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**MIE** stands for machine interrupt enable
Enable machine interrupts

08002914 <main>:
    ...
    csrr a5,mstatus  # read CSR mstatus to a5
    ori a5,a5,8     # set bit#3 of a5 to 1
    csrw mstatus,a5 # write CSR mstatus
    ...

int main() {
    ...
    int mstatus;
    asm("csrr %0, mstatus" : "=r"(mstatus));
    asm("csrw mstatus, %0" ::"r"(mstatus | 0x8));
    ...
}
Another CSR mie (not mstatus)

- mstatus.MIE is bit #3 in mstatus
- mie is another CSR, and mie.MTIE is bit #7 in mie
Enable timer interrupt

08002914 <main>:
.
  csrr a5,mie # read CSR mie to a5
  ori a5,a5,128 # set bit#7 of a5 to 1
  csrw mie,a5 # write CSR mie
.
int main() {
  int mie;
  asm("csrr %0, mie" : "=r"(mie));
  asm("csrw mie, %0" ::"r"(mie | 0x80));
  ...
}
int main() {
    .
    int mstatus, mie;
    asm("csrr %0, mstatus" : "=r"(mstatus));
    asm("csrw mstatus, %0" ::"r"(mstatus | 0x8));
    asm("csrr %0, mie" : "=r"(mie));
    asm("csrw mie, %0" ::"r"(mie | 0x80));
    .
}
Summary of timer interrupt

• How to register an interrupt handler?
  • write the address of handler() to mtvec

• How to set a timer?
  • write (mtime + quantum) to mtimecmp

• How to enable timer interrupt?
  • set bit#3 of CSR mstatus and bit#7 of CSR mie
CSR is a key CPU support for OS

• How to register an interrupt handler?
  • write the address of handler() to mtvec

• How to set a timer?
  • write (mtime + quantum) to mtimecmp

• How to enable timer interrupt?
  • set bit#3 of CSR mstatus and bit#7 of CSR mie
int quantum = 50000;

void handler() {
    earth->tty_info("Got timer interrupt.");
    mtimecmp_set(mtime_get() + quantum);
}

int main() {
    earth->tty_success("A timer interrupt example.");

    asm("csrw mtvec, %0" ::"r"(handler));
    mtimecmp_set(mtime_get() + quantum);

    int mstatus, mie;
    asm("csrr %0, mstatus" : "=r"(mstatus));
    asm("csrw mstatus, %0" ::"r"(mstatus | 0x8));
    asm("csrr %0, mie" : "=r"(mie));
    asm("csrw mie, %0" ::"r"(mie | 0x80));

    while(1);
}
demo code in microSD card

demo code in microSD card

earth layer in boot ROM containing
1. SD card driver for loading demo
2. TTY driver for printing to screen

https://github.com/yhzhang0128/egos-2000/tree/timer_demo/grass
## Timer is interrupt #7

### Interrupt Exception Codes

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0–2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Machine software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>4–6</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>Machine timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>8–10</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>Machine external interrupt</td>
</tr>
<tr>
<td>1</td>
<td>≥ 12</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Instruction address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Instruction access fault</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>Load address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>Load access fault</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>Store/AMO address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>Store/AMO access fault</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>Environment call from U-mode</td>
</tr>
<tr>
<td>0</td>
<td>9–10</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>Environment call from M-mode</td>
</tr>
<tr>
<td>0</td>
<td>≥ 12</td>
<td>Reserved</td>
</tr>
<tr>
<td>Interrupt</td>
<td>Exception Code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>1</td>
<td>0–2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Machine software interrupt</td>
</tr>
<tr>
<td>1</td>
<td>4–6</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>Machine timer interrupt</td>
</tr>
<tr>
<td>1</td>
<td>8–10</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>Machine external interrupt</td>
</tr>
<tr>
<td>1</td>
<td>≥ 12</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Instruction address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Instruction access fault</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>Load address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>Load access fault</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>Store/AMO address misaligned</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>Store/AMO access fault</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>Environment call from U-mode</td>
</tr>
<tr>
<td>0</td>
<td>9–10</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>Environment call from M-mode</td>
</tr>
<tr>
<td>0</td>
<td>≥ 12</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
void kernel() {
    // registered to CSR mtvec
    int mcause;
    __asm__ volatile("csrr %0, mcause" : "=r"(mcause));

    int id = mcause & 0x3ff;
    if (mcause & (1 << 31)) {
        if (id == 7) { yield(); }
    } else {
        if (id == 8) { syscall_handler(); }
        else { fault_handler(); }
    }
}
Kernel $\approx$ timer handler + system call handler + fault handler
Design of projects P1 and P2

```c
void kernel() {
    int mcause;
    __asm__ volatile("csrr %0, mcause" : "=r"(mcause));

    int id = mcause & 0x3ff;
    if (mcause & (1 << 31)) {
        // P1: multi-threading
        if (id == 7) { yield(); }  
    } else {
        // P2: system call and memory protection
        if (id == 8) { syscall_handler(); }  
        else { fault_handler(); }  
    }
}
```
Some details: **memory-mapped register**

<table>
<thead>
<tr>
<th>Address</th>
<th>Width</th>
<th>Attr.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2000000</td>
<td>4B</td>
<td>RW</td>
<td>msip for hart 0</td>
</tr>
<tr>
<td>0x2004008</td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x200bff7</td>
<td></td>
<td></td>
<td>mtimecmp for hart 0</td>
</tr>
<tr>
<td>0x2004000</td>
<td>8B</td>
<td>RW</td>
<td></td>
</tr>
<tr>
<td>0x2004008</td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x200bff7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x200bff8</td>
<td>8B</td>
<td>RW</td>
<td>mtime</td>
</tr>
<tr>
<td>0x200c000</td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
</tbody>
</table>

mtime_get() reads 8 bytes from
mtimecmp_set() writes 8 bytes to

Page 44 of Sifive FE310 manual, v19p04
Homework

• **P2** has been released and it is due on **Oct 20**.

• Read the **4 files** of the timer demo program.
  