Interrupt and Exception
High-level roadmap

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

- [ basic RISC-V CPU ] user-level threading
- [ + timer interrupt ] timeshare threading
- [ + ecall exception ] system call
- [ + privilege levels ] memory protection
- [ + I/O bus ] disk driver, file systems and cache
Set a timer
void handler() {
    earth->tty_info("Got timer interrupt.");
    // set a timer
}

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

    while(1);
}
Execution of the program

void handler() {
    earth->tty_info("...");
    // set a timer
}

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

    while(1);
}

When timer is running

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

• How to set 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

![Diagram of mtvec CSR]

<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><em>Reserved</em></td>
</tr>
</tbody>
</table>

Table 3.5: Encoding of mtvec MODE field.
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
  . . .
Register an interrupt handler

```c
void handler() {
    ...
}

int main() {
    /* Register handler with direct mode */
    asm("csrw mtvec, %0" ::"r"(handler));
    ...
}
```
• How to **register handler()** as interrupt handler?

  ➡️ How to **set** 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 > mtimecmp

CLINT

Machine time (mtime)

Machine time compare (mtimecmp)

E31
A RISC-V core

Page 38 of Sifive FE310 manual, v19p04
int quantum = 50000; // clock cycles

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

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

Balloon How to enable timer interrupt?
# The mstatus CSR

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>WPRI</td>
<td>TSR</td>
<td>TW</td>
<td>TVM</td>
<td>MXR</td>
<td>SUM</td>
<td>MPRV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**MIE** stands for machine interrupt enable

Page 20 of RISC-V manual, volume 2, v1.10
Enable machine interrupts

08002914 <main>:
...  
csrr a5,mstatus  # read CSR mstatus to a5
ori a5,a5,8   # set bit3 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.MIE`)

- `mstatus.MIE` is bit #3 in `mstatus`
- `mie` is another 32-bit CSR, and `mie.MTIE` is bit #7 in `mie`

**MTIE** stands for machine timer interrupt enable

<table>
<thead>
<tr>
<th>XLEN-12</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WPR1</td>
<td>MEIE</td>
<td>WPRI</td>
<td>SEIE</td>
<td>UEIE</td>
<td>MTIE</td>
<td>WPRI</td>
<td>STIE</td>
<td>UTIE</td>
<td>MSIE</td>
<td>WPRI</td>
<td>SSIE</td>
<td>USIE</td>
<td></td>
</tr>
<tr>
<td>XLEN-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Enable timer interrupt

08002914 <main>:

    .
    .
    csrr a5,mie # read CSR mie to a5
    ori a5,a5,128 # set bit7 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));
    .
}
Altogether: Enable timer interrupt

```c
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 function handler() to mtvec

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

• How to enable timer interrupt?
  • set bit#3 of mstatus and bit#7 of mie to 1
CSR is an important CPU support for OS

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

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

• How to enable timer interrupt?
  • set bit#3 of mstatus and bit#7 of mie to 1
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);
    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

earth layer code in boot ROM
1. Load demo from microSD
2. Print strings to the screen

https://github.com/yhzhang0128/egos-2000/tree/timer_example1/grass
## Timer is interrupt #7

### Interrupts

### Exceptions

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Exception Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>

### 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>
</tbody>
</table>
System call is exception #8, #11

<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>
</tbody>
</table>
Kernel $\approx$ timer handler + system call handler + fault handler
Kernel ≈ 3 handlers

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(); }
            }
        } else {
            yield();
        }
    }
Design of the 4411 projects

```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>0x200b7f</td>
<td>8B</td>
<td>RW</td>
<td>mtimecmp for hart 0</td>
</tr>
<tr>
<td>0x2004000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2004008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x200b7f</td>
<td>8B</td>
<td>RW</td>
<td>mtime</td>
</tr>
<tr>
<td>0x200b7f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x200c000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Page 44 of Sifive FE310 manual, v19p04
Some details: mtime rollover

When reading the lower 4 bytes, mtime is 0x00000000 0xffffffff

When reading the higher 4 bytes, mtime is 0x00000001 0x00000000

Combine the two 0x00000001ffffffff is wrong!

mtime automatically increments!
Some details: mtime rollover

long long mtime_get() {
    int low, high;
    do {
        high = *(int*)(0x200bff8 + 4);
        low  = *(int*)(0x200bff8);
    } while ( *(int*)(0x200bff8 + 4) != high );
    return (((long long)high) << 32) | low;
}

void mtimecmp_set(long long time) {
    *(int*)(0x2004000 + 4) = 0xFFFFFFFF;
    *(int*)(0x2004000 + 0) = (int)time;
    *(int*)(0x2004000 + 4) = (int)(time >> 32);
}
Homework

• P2 has been released and it is due on Mar 24.
• Read the 4 files of the timer handler program.
  • https://github.com/yhzhang0128/egos-2000/tree/timer_example1/grass