

Week 4.a

CS6640

09/26 2023

<https://naizhengtan.github.io/23fall/>

1. Timer interrupt overview ↙
2. Register interrupt handler
3. Periodical timer interrupt
4. Enabling timer
5. Interrupt in egos-2k+



Roadmap of lab2--lab4

[basic RISC-V CPU] non-preemptive multi-threading (lab2)
[+ timer interrupt] preemptive scheduling (lab3)
[+ privilege levels] memory protection and syscall (lab4)

CPU view: Core-local Interrupt (CLINT)

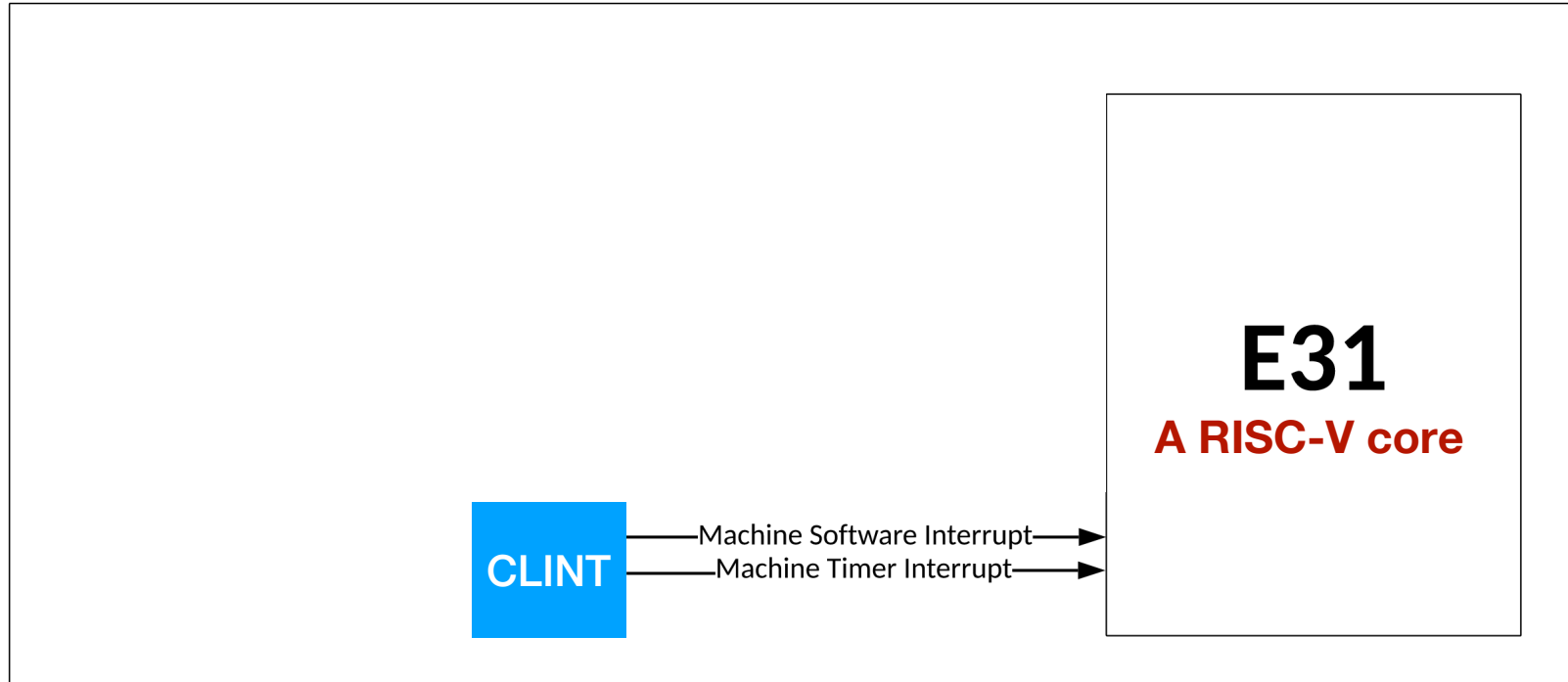


Figure 4 of Sifive FE310 manual

CS6640 Week4.a

1. Timer interrupt handler

```

void handler() {
    CRITICAL("Got a timer interrupt!");
    // (4) reset timer

}

int main() {
    CRITICAL("This is a simple timer example");

    // (1) register handler() as interrupt handler

    // (2) set a timer

    // (3) enable timer interrupt

    while(1);
}

```

2. Background: RISC-V assembly II

Assembler instructions with C expression operands:

⇒ **asm(Template : OutputOperands : InputOperands)**

a) Template: a string that is the template for the assembler code.

```
asm("mret");
```

b) OutputOperands: the C variables modified by the instructions in the Template.

```

void *sp;
asm("mv %0, sp" : "=r" sp);

```

Handwritten annotations: "var" above *sp, "register" above "=r", "modified" below sp, and arrows pointing from the annotations to the corresponding parts of the code.

c) InputOperands: C expressions read by the instructions in the Template.

```

int mie;
asm("csrr %0, mie" : "=r" (mie));
asm("csw mie, %0" : "=r" (mie));

```

Handwritten annotations: "var" above mie in both lines, and "modified" above mie in the second line, with arrows pointing to the mie expressions.

CSP: mtvec ← handler

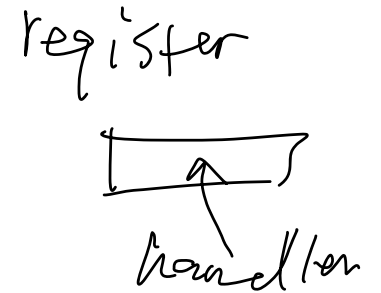
addr

A programmer's view

```
void handler() {  
    printf("Got a timer interrupt!");  
    // (4) reset timer  
}
```

```
int main() {  
    // (1) register interrupt handler  
    // (2) set a timer  
    // (3) enable timer interrupt  
  
    while(1);  
}
```

Call foo *addr*



The mtvec CSR

"mstatus"
"MPP"

bit31

bit2

bit1

bit0



Value	Name	Description
0	Direct	All exceptions set pc to BASE.
1	Vectored	Asynchronous interrupts set pc to BASE+4×cause.
≥2	—	Reserved

Table 3.5: Encoding of mtvec MODE field.

A programmer's view

```
void handler() {  
    printf("Got a timer interrupt!");  
    // (4) reset timer  
}
```

→ mtime

```
int main() {  
    // (1) register interrupt handler  
    // (2) set a timer + 1 QUANTUM  
    // (3) enable timer interrupt  
  
    while(1);  
}
```

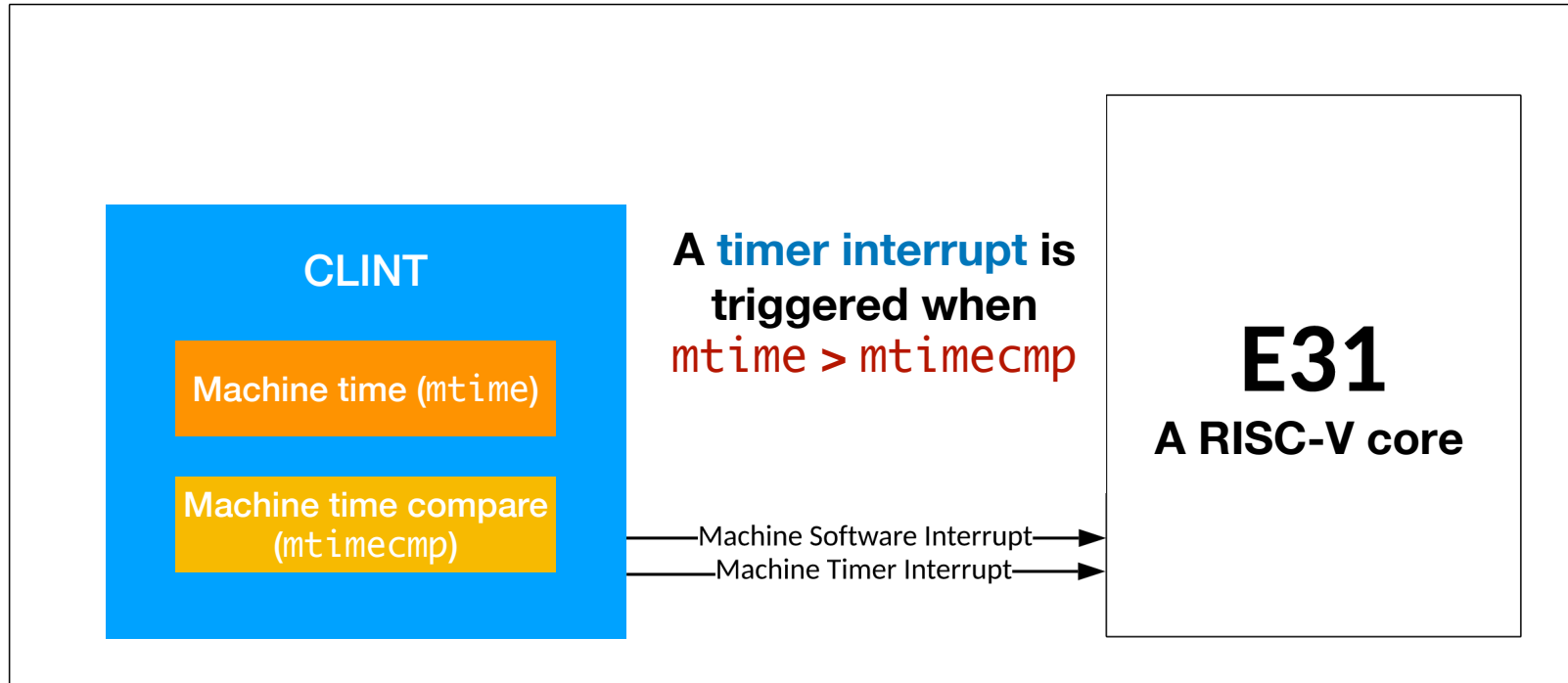
→ mtimecmp

←

- get current time
- set mtimecmp to


current time
+ 1Q


The `mtime` and `mtimecmp` CSRs



CLINT CSRs are **memory-mapped**

Address	Width	Attr.	Description
0x20000000	4B	RW	msip for hart 0
0x2004008			Reserved
...			
0x200bff7			
0x2004000	8B	RW	mtimecmp for hart 0
0x2004008			Reserved
...			
0x200bff7			
0x200bff8	8B	RW	mtime
0x200c000			Reserved

`mtimecmp_set()` writes 8 bytes to 

`mtime_get()` reads 8 bytes from 

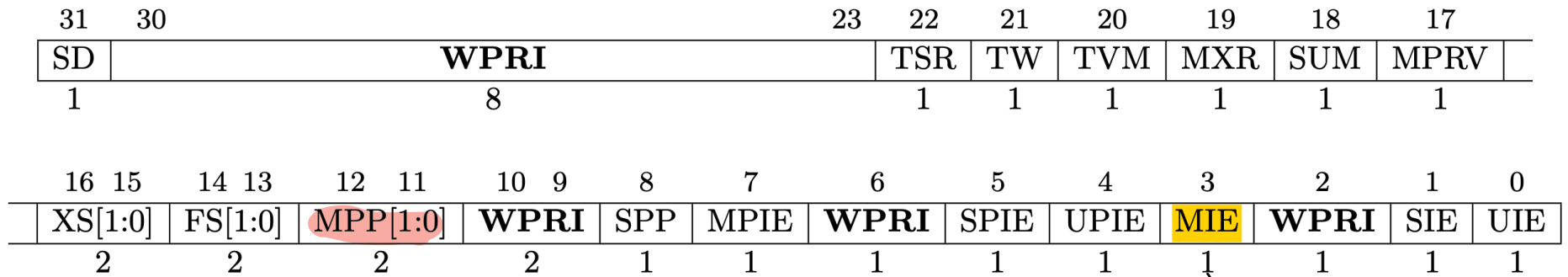
A programmer's view

```
void handler() {  
    printf("Got a timer interrupt!");  
    // (4) reset timer  
}
```

```
int main() {  
    // (1) register interrupt handler  
    // (2) set a timer  
    // (3) enable timer interrupt  
  
    while(1);  
}
```



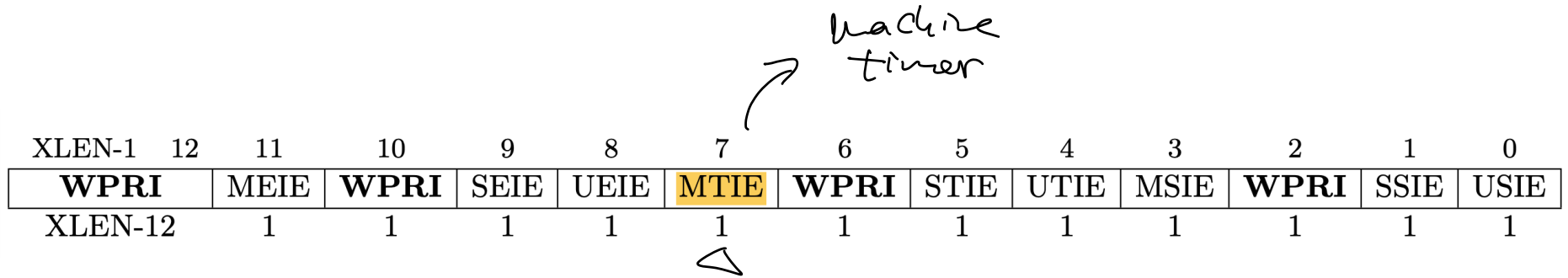
The **mstatus** CSR



MIE stands for machine interrupt enable

↳ 0: disabled
 ↑: enabled

The **mie** CSR (not mstatus.MIE)



MTIE stands for machine timer interrupt enable

Next...

- two bummers in setting timers
- interrupts beyond CLINT
- RISC-V fine-grained control of interrupts
- how to know which interrupt is triggered?

a bummer: mtime rollover

Higher 4 bytes

Lower 4 bytes

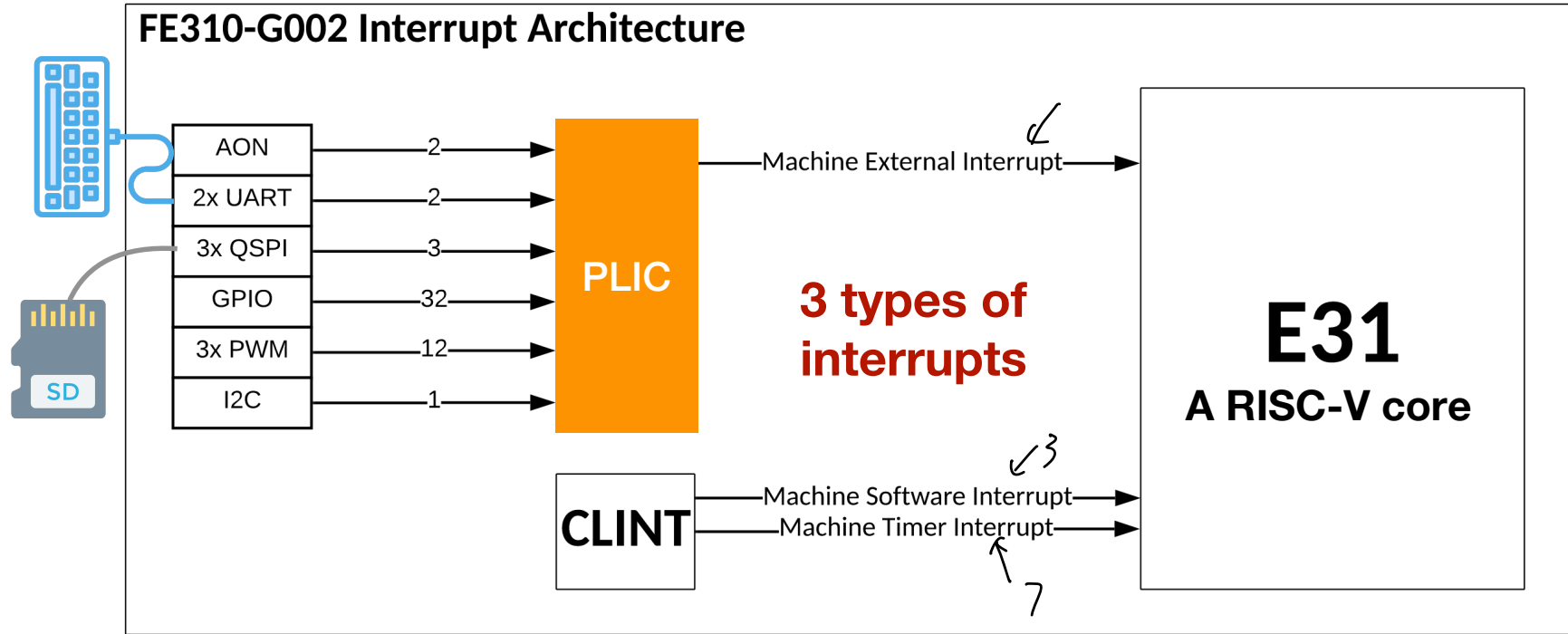
When reading the **lower** 4 bytes, mtime is
`0x00000000` `0xffffffff`

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

Combine the two `0x00000001ffffffff` is wrong!

There is another bummer: mtime overflow

Platform Interrupt Controller (PLIC)



mie provides fine-grained control

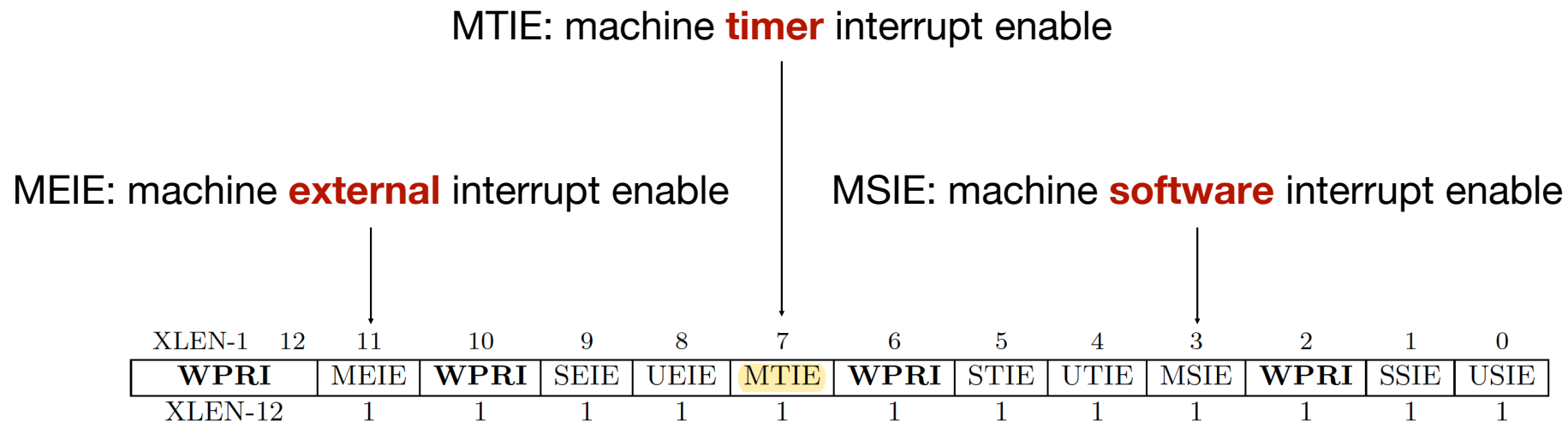


Figure 3.12: Machine interrupt-enable register (mie).

Timer is interrupt #7

Interrupts

Exceptions

Interrupt	Exception Code	Description
1	0	<i>Reserved</i>
1	1	Supervisor software interrupt
1	2	<i>Reserved</i>
1	3	Machine software interrupt
1	4	<i>Reserved</i>
1	5	Supervisor timer interrupt
1	6	<i>Reserved</i>
1	7	Machine timer interrupt
1	8	<i>Reserved</i>
1	9	Supervisor external interrupt
1	10	<i>Reserved</i>
1	11	Machine external interrupt
1	12-15	<i>Reserved</i>
1	≥ 16	<i>Designated for platform use</i>
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	<i>Reserved</i>
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	<i>Reserved</i>
0	15	Store/AMO page fault
0	16-23	<i>Reserved</i>
0	24-31	<i>Designated for custom use</i>
0	32-47	<i>Reserved</i>
0	48-63	<i>Designated for custom use</i>
0	≥ 64	<i>Reserved</i>

"in cause"



System calls?

Interrupts

Exceptions

Interrupt	Exception Code	Description
1	0	<i>Reserved</i>
1	1	Supervisor software interrupt
1	2	<i>Reserved</i>
1	3	Machine software interrupt
1	4	<i>Reserved</i>
1	5	Supervisor timer interrupt
1	6	<i>Reserved</i>
1	7	Machine timer interrupt
1	8	<i>Reserved</i>
1	9	Supervisor external interrupt
1	10	<i>Reserved</i>
1	11	Machine external interrupt
1	12-15	<i>Reserved</i>
1	≥ 16	<i>Designated for platform use</i>
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	<i>Reserved</i>
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	<i>Reserved</i>
0	15	Store/AMO page fault
0	16-23	<i>Reserved</i>
0	24-31	<i>Designated for custom use</i>
0	32-47	<i>Reserved</i>
0	48-63	<i>Designated for custom use</i>
0	≥ 64	<i>Reserved</i>



ECall

