Week 4.b
CS6640
01/30 2026
https://naizhengtan.github.io/26spring/

- [x] 1. egos-2k+ booting process
- [ ] 2. RISC-V assembly in C (15min)
- [ ] 3. Timer interrupt (30min)

Q: why did a process finish already?

Booting
---

* in earth/boot.c

  Q: what does asm("csrr %0, mhartid" : "=r"(core_id)) do?

* in grass/init.c

  Q: Does it work to directly jump to APP_ENTRY? (instead of using 'mret')

* in sys_shell.c

  [answering the first question]

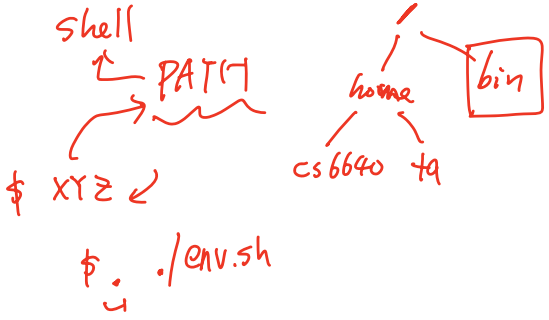
"cd"

CS6640, Week 04a

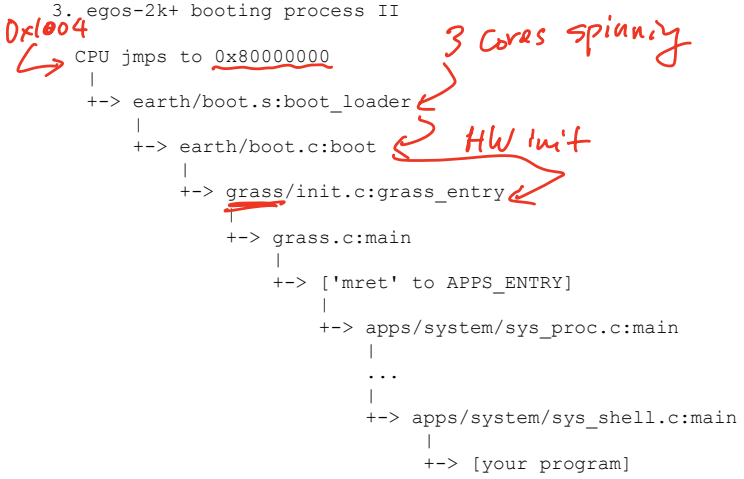
1. egos architecture
[…]

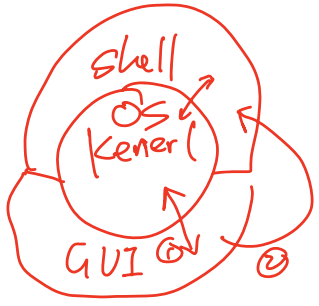
2. egos-2k+ booting process I

```
1   -------- Simulate on QEMU-RISCV --------
2   qemu-system-riscv32 -M virt -smp 4 -m 8M -bios tools/egos.bin -nographic
3      -drive if=pflash,format=raw,unit=1,file=tools/qemuROM.bin
4      -device sdhci-pci,addr=0x1
5      -device sd-card,drive=MMC
6      -drive if=none,file=tools/disk.img,format=raw,id=MMC
7   [CRITICAL] --- Booting on QEMU with core #2 ---
8   [SUCCESS] Finished initializing the tty and disk devices
9   [INFO] Use direct mode and put the address of the trap_entry into mtvec
10  [SUCCESS] Finished initializing the MMU, timer and interrupts
11  [SUCCESS] Enter the grass layer
12  [INFO] Load kernel process #1: sys_process
13  [INFO] Load 0x4400 bytes to 0x80200000
14  [INFO] Load 0x510 bytes to 0x80208000
15  [SUCCESS] Enter kernel process GPID_PROCESS
16  [INFO] Load kernel process #2: sys_terminal
17  [INFO] Load 0x3660 bytes to 0x80200000
18  [INFO] Load 0x274 bytes to 0x80208000
19  [SUCCESS] Enter kernel process GPID_TERMINAL
20  [INFO] sys_process receives: Finish GPID_TERMINAL initialization
21  [INFO] Load kernel process #3: sys_file
22  [INFO] Load 0x5040 bytes to 0x80200000
23  [INFO] Load 0x9c4 bytes to 0x80208000
24  [SUCCESS] Enter kernel process GPID_FILE
25  [INFO] sys_process receives: Finish GPID_FILE initialization
26  [INFO] Load kernel process #4: sys_shell
27  [INFO] Load 0x4da4 bytes to 0x80200000
28  [INFO] Load 0x89c bytes to 0x80208000
29  [CRITICAL] Welcome to the egos-2k+ shell!
30  → /home/cs6640 %
```



3. egos-2k+ booting process II





OSI Week4.b

1. 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("lw ra,12(sp)");
asm("ret");
```

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

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

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

```
void *sp = (void*)0x803fffc0;
asm("mv sp,%0" : : "r"(sp));
```

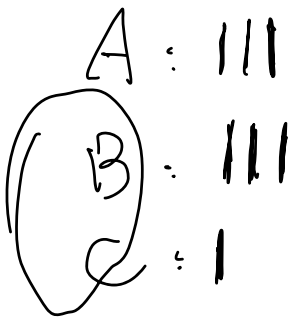
[read more: https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html]

2. 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);
}
```

Q: if you were a CPU designer, how would you like to define the interrupt interface?

CPU:
(A) instructions
(B) registers
(C) memory addresses



Timer interrupts

[borrowed from Yunhao's
CS 4411/5411: Practicum in Operating Systems, 22fall]

CPU view: Core-local Interrupt (CLINT)

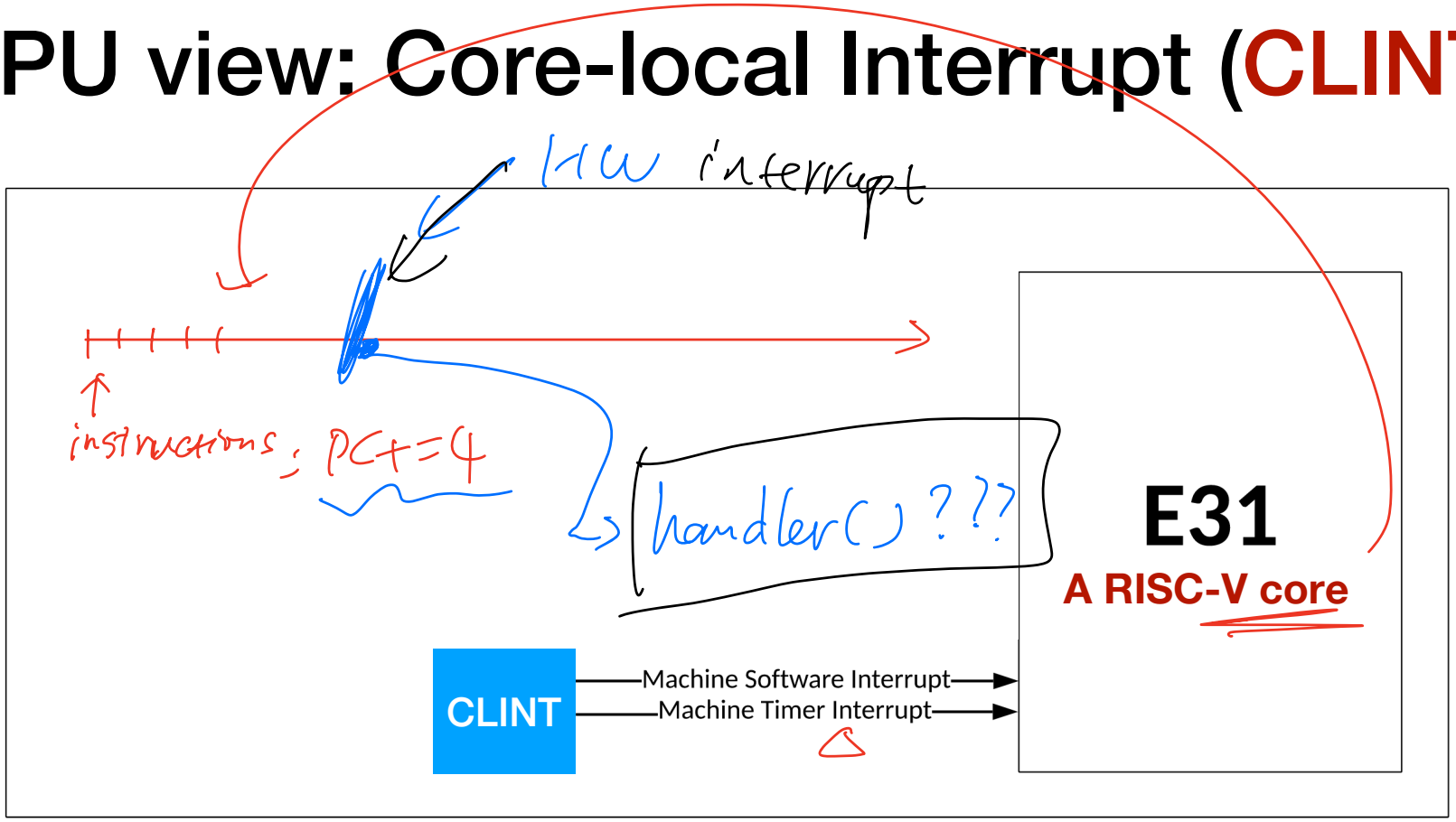


Figure 4 of Sifive FE310 manual

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 `mtvec` CSR

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

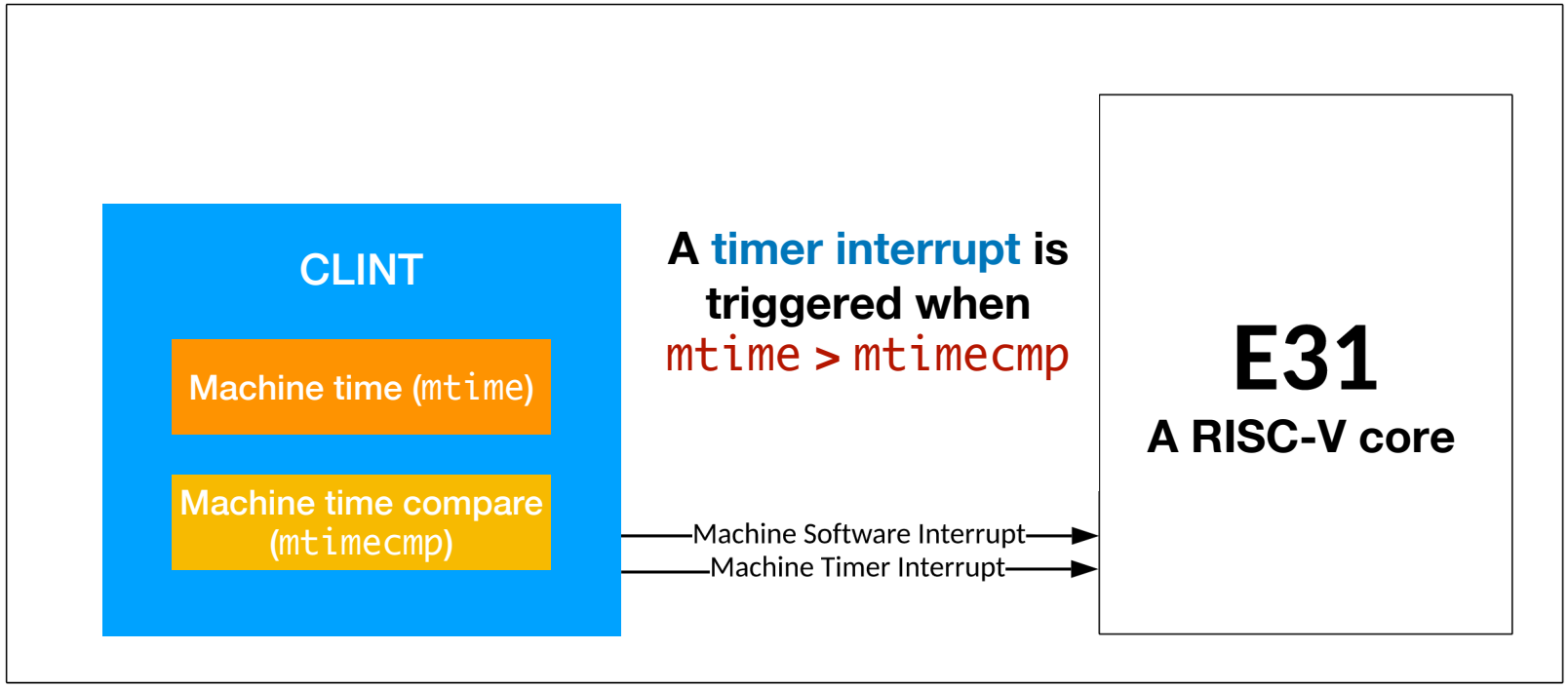
Table 3.5: Encoding of `mtvec` MODE field.

RISC-V privileged spec

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 mtime and mtimecmp CSRs



Sifive FE310 manual

CLINT CSRs are memory-mapped

`mtimecmp_set()` writes 8 bytes to

`mtime_get()` reads 8 bytes from

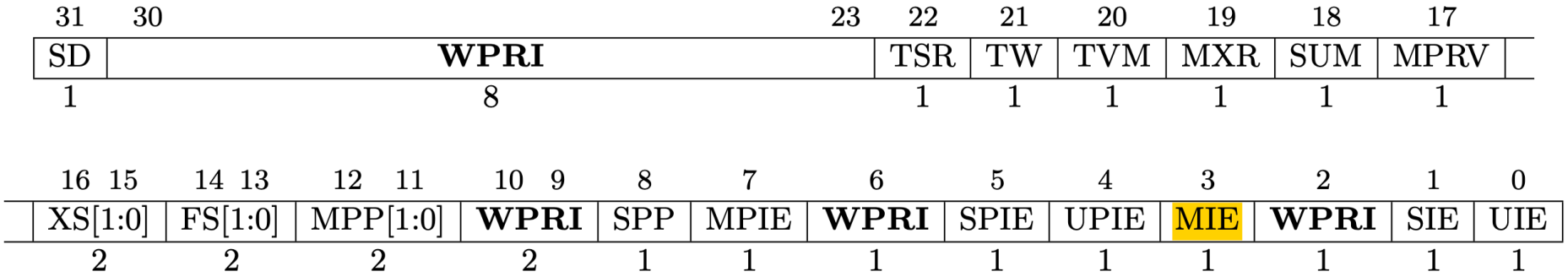
Address	Width	Attr.	Description
0x2000000	4B	RW	msip for hart 0
0x2004008			Reserved
…			
0x200bff7			
0x2004000	8B	RW	mtimecmp for hart 0
0x2004008			Reserved
…			
0x200bff7			
0x200bff8	8B	RW	mtime
0x200c000			Reserved

Sifive FE310 manual

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



31	30 … 23	22	21	20	19	18	17
SD	WPRI	TSR	TW	TVM	MXR	SUM	MPRV
1	8	1	1	1	1	1	1

16 15	14 13	12 11	10 9	8	7	6	5	4	3	2	1	0
XS[1:0]	FS[1:0]	MPP[1:0]	WPRI	SPP	MPIE	WPRI	SPIE	UPIE	MIE	WPRI	SIE	UIE
2	2	2	2	1	1	1	1	1	1	1	1	1

MIE stands for machine interrupt enable

RISC-V privileged spec

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

XLEN-1 … 12	11	10	9	8	7	6	5	4	3	2	1	0
WPRI	MEIE	**WPRI**	SEIE	UEIE	MTIE	**WPRI**	STIE	UTIE	MSIE	**WPRI**	SSIE	USIE
XLEN-12	1	1	1	1	1	1	1	1	1	1	1	1

MTIE stands for machine timer interrupt enable

RISC-V privileged spec