Lab1: 机器启动

> author：魏新鹏
>
> student ID：519021910888

1 思考题 1

阅读 _start 函数的开头，尝试说明 ChCore 是如何让其中一个核首先进入初始化流程，并让其他核暂停执行的。

```
mrs x8, mpidr_el1
and x8, x8, #0xFF
cbz x8, primary

/* hang all secondary processors before we introduce smp */
b    .
```

From arm Developer, mpidr_el1系统寄存器的0~7位会存储

> Affinity level 0. The level identifies individual threads within a multithreaded core.

因此只有0号cpu的x8才是0，因此才会跳转到primary继续执行，其他cpu咋会在`b .`（跳转到当前处）loop，从而停止进一步执行。

2 练习题 2

在 arm64_elX_to_el1 函数的 LAB 1 TODO 1 处填写一行汇编代码，获取 CPU 当前异常级别。

`mrs x9, currentel`

currentel system register保存了当前的异常等级。mrs将系统寄存器move到通用寄存器。

3 练习题 3

在 arm64_elX_to_el1 函数的 LAB 1 TODO 2 处填写大约 4 行汇编代码，设置从 EL3 跳转到EL1 所需的 elr_el3 和 spsr_el3 寄存器值。具体地，我们需要在跳转到 EL1 时暂时屏蔽所有中断、并使用内核栈（ sp_el1 寄存器指定的栈指针）。

```
adr x9, .Ltarget
msr elr_el3, x9 #设置返回地址
mov x9, SPSR_ELX_DAIF | SPSR_ELX_EL1H #屏蔽中断DAIF，设置exception level和栈指针 EL1H
msr spsr_el3, x9
```

> ref

4 思考题 4

结合此前 ICS 课的知识，并参考 kernel.img 的反汇编（通过 aarch64-linux-gnu-objdump -S 可获得），说明为什么要在进入 C 函数之前设置启动栈。如果不设置，会发生什么？

```
stp x29, x30, [sp, #-16]! //first line of init_t
```

因为进入init_c后，函数会保存stack pointer和link pointer以变结束后恢复并返回至原函数。这些寄存器都是保存在栈上的，如果没有初始化sp，那么sp中就是一个随机值，往未知的的地址存数据会导致UNDEFINED behavior或者带来错误。

5 思考题 5

在实验 1 中，其实不调用 clear_bss 也不影响内核的执行，请思考不清理 .bss 段在之后的何种情况下会导致内核无法工作。

正常情况下bss段被加载到内存中时会由OS初始化成0，但因为我们是OS，所以就需要我们自己初始化。

而bss段代表未初始化的静态和全局变量，c默认是将其全部初始化为0，如果实际上没有初始化为了0，那么这些静态变量和全局变量就会带有随机的初始值，这会导致UNDEFINED behavior或者错误。

6 练习题 6

在 kernel/arch/aarch64/boot/raspi3/peripherals/uart.c 中 LAB 1 TODO 3 处实现通过 UART 输出字符串的逻辑。

```
void uart_send_string(char *str)
{
	/* LAB 1 TODO 3 BEGIN */
	for (; *str != '\0'; str++) {
		early_uart_send(*str);
	}
	/* LAB 1 TODO 3 END */
}
```

7 练习题 7

在 kernel/arch/aarch64/boot/raspi3/init/tools.S 中 LAB 1 TODO 4 处填写一行汇编代码，以启用 MMU。

```
orr     x8, x8, #SCTLR_EL1_M
```

设置启动MMU