Lab1-Document

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。

```
BEGIN_FUNC(_start)
    mrs x8, mpidr_el1
    and x8, x8, #0xFF
    cbz x8, primary

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

primary:
    /* Turn to el1 from other exception levels. */
    bl  arm64_elX_to_el1
```

根据上述代码可以看出，`ChCore`先将CPU的core ID 移到`x8`寄存器中，紧接着将其与`0xFF`做`and`操作并将结果存放在`x8`寄存器中。接下里，`x8`寄存器里的值与0做比较，若相等，则说明是`CPU0`，接下来跳转到primary部分进入初始化流程；若不等，说明是其他core ID不为0的CPU，它们将会挂起暂缓执行。

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。

通过`CurrentEL`系统寄存器可获得当前异常级别，用`mrs`移到`x9`寄存器中，此时可以用gdb调试查看异常级别

```
/* LAB 1 TODO 1 BEGIN */
mrs x9, CurrentEL
/* LAB 1 TODO 1 END */
```

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）。

在`elr_el3`中将`.Ltarget`处设置为`EL3`的返回地址

设置`EL3`的状态寄存器`spsr_el3`，其中`SPSR_ELX_DAIF`的含义为`D: debug; A: error; I: interrupt; F: fast interrupt`，可以暂时屏蔽所有中断，而`SPSR_ELX_EL1H`则指定了使用的内核栈为`sp_el1`寄存器指定的栈指针

```
/* LAB 1 TODO 2 BEGIN */
adr x9, .Ltarget
msr elr_el3, x9
mov x9, SPSR_ELX_DAIF | SPSR_ELX_EL1H
msr spsr_el3, x9
/* LAB 1 TODO 2 END */
```

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?

因为C函数调用会把局部变量和一些参数放在栈上。如果不进行设置，`sp`的初始值为0，而栈是向下增长的，地址会变为负数，这就会导致内存非法访问，没办法在合适的内存区域进行函数调用，也就没法正常执行C函数`init_c`。

```
0000000000080010 <primary>:
   80010:   94001ffc    bl 88000 <arm64_elX_to_el1>
   80014:   580000a0    ldr   x0, 80028 <primary+0x18>
   80018:   91400400    add   x0, x0, #0x1, lsl #12
   8001c:   9100001f    mov   sp, x0
   80020:   940020e2    bl 883a8 <init_c>
```

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。

`.bss`段存放的是未初始化的全局变量和静态变量，在加载时需要初始化为 0。如果这些变量在上一次调用中被修改并在`.bss`中被保存了下来，而之后重新运行代码调用时没有被清除，可能使得遇到代码遇到异常的初始值（非0），导致内核无法正常工作

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。

通过循环调用early_uart_send()来将字符串里的字符一个个发送出去，进而达到发送出整个字符串的效果。

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

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。

通过`orr`操作按位取或，将系统寄存器`sctlr_el1`的`M`字段置为1，表示启用`MMU`

```
/* LAB 1 TODO 4 BEGIN */
orr     x8, x8, #SCTLR_EL1_M
/* LAB 1 TODO 4 END */
```

519021910594

陶昱丞