实验 1:机器启动 实验报告

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)
    /*MPIDR_EL1是多核标志寄存器,存储cpuid,mrs表示将其移动到x8寄存器*/
    mrs x8, mpidr_el1
    /*将cpuid和0xFF做and, 保留x8的低8位*/
    and x8, x8, #0xFF
    /*cbz表示为0则跳转,如果x8为0则为0号CPU核,跳转到primary段,使第一个核进入初始化流程*/
    cbz x8, primary

    /* hang all secondary processors before we introduce smp */
    /*b指令表示跳转, b .表示跳转到本行, 使得除了0号核之外的其余核hang,等待0号核完成初始化*/
    b   .
    /*0号核进入初始化流程*/
primary:
    /*0号核的初始化流程*/
```

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
```

使用mrs将系统寄存器`CurrentEL`中存储的异常级别放入`x9`寄存器中

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）。

```
mov x9, SPSR_ELX_EL1H | SPSR_ELX_DAIF
msr spsr_el3, x9
/*以上两句, 用spsr_el3寄存器保存切换时的程序状态(包括异常返回后的异常级别),便于返回后恢复*/
adr x9, .Ltarget
msr elr_el3, x9
/*以上两句, 用elr_el3寄存器保存.Ltarget的地址作为return address使用*/
```

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?

```
0000000000088398 <init_c>:
   88398:   a9bf7bfd    stp x29, x30, [sp, #-16]!
   8839c:   910003fd    mov x29, sp
   883a0:   97ffffe5    bl  88334 <clear_bss>
```

（假设sp指针在CPU启动后初始化为0），`init_c`有压栈操作（`stp x29, x30, [sp, #-16]!`），如果不提前设置sp则`sp-16`为负数，无法运行;

即使sp在CPU启动后初始值不为0在之后的使用中仍有可能因为栈的向下使用被耗尽空间使得系统无法继续运行，或sp初始值超过内存物理地址上限导致访存错误

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段用于存放未初始化或初始化为0的全局变量和静态局部变量, 硬件启动后这段内存的初始值可能不为0，如果没有使用`clear_bss`将其初始化为0可能导致程序运行时bss段存放的变量初始值不为0而引发错误

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) {
  char* ptr = str;
  while (*ptr != '\0') {
    early_uart_send(*ptr);
    ptr++;
  }
}
```

思路为遍历`str`, 将str中的每个字符使用`early_uart_send`输出直到遇到`\0`(表示字符串结束)

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
```

意为`x8 = x8 | #SCTLR_EL1_M`, 将x8中SCTLR_EL1_M位设为1来启动MMU(之后`msr sctlr_el1, x8`将`sctlr_el1`的值设为`x8`的值后生效)