1 思考题

start 函数的开头如下。在所有 CPU 核心各自调用 start 函数后，系统会对所有核分别通过 mpidr_el1 寄存器1提取对应核心的 id，之后在第 10 行提取其低 8 位，存在寄存器 x8 中。只有当前的 cpu 核是 0 号核时，cbz 指令才能够发生跳转，进入下面的 primary 流程执行接下来的操作。而所有并非 0 号核的 cpu 未发生跳转，会来到第 14 行的无条件跳转处，并在这一行无限循环，以达到一种暂停的效果。

```
9     mrs x8, mpidr_el1
10    and x8, x8, #0xFF
11    cbz x8, primary
12
13    /* hang all secondary processors before we introduce smp */
14    b   .
```

AArch64 System Register Descriptions
D13.2 General system control registers

D13.2.101 MPIDR_EL1, Multiprocessor Affinity Register

The MPIDR_EL1 characteristics are:

Purpose

In a multiprocessor system, provides an additional PE identification mechanism for scheduling purposes.

Configurations

AArch64 System register MPIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MPIDR[31:0].

In a uniprocessor system, Arm recommends that each Aff<n> field of this register returns a value of 0.

Attributes

MPIDR_EL1 is a 64-bit register.

图 1: mpidr_el1 寄存器说明

2 练习题

根据提示，CurrentEL 系统寄存器中已经存储了当前的异常级别，通过阅读下文代码我们知道，后续会将 x9 寄存器中的内容和预定义的常量 CURRENTEL_EL1 等进行比较以判断是否发生跳转，故我们知道应该把当前的异常级别存入寄存器 x9。同时，为了能够在系统寄存器和通用寄存器之间转移数据，我们要使用 mrs 指令。所给出的一行代码如下。

```
59    /* LAB 1 TODO 1 BEGIN */
60    mrs x9, CurrentEL
```

```
61    /* LAB 1 TODO 1 END */
```

3 练习题

根据 arm 官方 isa 的解释2我们可以知道，在调用 eret 指令时，系统会根据当前的 EL，查询当前 EL 所对应的 ELR 和 SPSR 的内容来进行返回。再结合文档中指出的，elr_elx 能够控制异常返回后执行的指令地址，spsr_elx 保存了返回后应恢复的程序状态，我们只需要在返回之前分别设定这两个系统寄存器的内容即可。

首先，由于我们是通过 start 函数的调用进入当前函数的，在本子进程结束后我们会回到 start 函数，也就是执行代码中.Ltarget 标签处所对应的 ret 指令，所以我们只需要先将其地址保存在寄存器 x9 中，再将 x9 内容保存到 elr_el3 中即可。而之所以要用两行汇编代码实现这一功能，据查询，是因为在 arm 指令集架构中，只有 msr 指令才能修改系统寄存器中的内容，所以这里 x9 寄存器只是临时帮我们保存一下数据。

其次，通过查阅下文第 121 行处为返回 EL2 设定寄存器的内容的汇编代码，我们可以知道，返回后的程序状态保存在 SPSR_ELX_DAIF, SPSR_ELX_EL1H 两个寄存器中。我们采取类似的操作，即可把程序状态保存在 spsr_el3 中。

```
77    /* LAB 1 TODO 2 BEGIN */
78    adr x9, . Ltarget
79    msr elr_el3, x9
80    mov x9, SPSR_ELX_DAIF | SPSR_ELX_EL1H
81    msr spsr_el3, x9
82    /* LAB 1 TODO 2 END */
```

D2.80 ERET

Returns from an exception. It restores the processor state based on SPSR_EL*n* and branches to ELR_EL*n*, where *n* is the current exception level..

Syntax

```
ERET
```

Usage

Exception Return using the ELR and SPSR for the current Exception level. When executed, the PE restores PSTATE from the SPSR, and branches to the address held in the ELR.

图 2: eret 指令说明

4 思考题

设置启动栈是为了给所调用的函数预留一定的空间，以供调用者和被调用者之间传递参数和返回值，并供被调用者保存自己在执行过程中所使用的一些不存储在寄存器中的局部变量，以及编译器生成的一些其他

的临时变量。

如果不设置，在被调用者使用的参数量较多，通用寄存器不够用的情况下，或者被调用者发生更复杂的函数调用的情况下，将没有一个很好的传参和返回值的机制，同时被调用者之间的寄存器值也会发生覆盖，导致计算上出现异常。

5 思考题

在当前默认的 c++ 标准下，全局变量与静态局部变量没有初始化值或初始化值为 0 时，都会放在.bss 段；初始化为非 0 值，则放在.data 段。如果我们不将.bss 段清零，那么当我们初始化了一个全局变量，并将其初始化为 0 时，实际上他的值会变成.bss 段对应位置在上次执行之后的结果，这样实际上该变量的值并不是我们所设定的 0。这样在我们后续用到此全局变量时，可能会引发一些意想不到的情况。

6 练习题

通过阅读上文函数 early_uart_send 我们知道，其功能是通过类似于轮询的方式，不断调用 early_uart_lsr 函数，确认 AUX_MU_LSR_REG 寄存器的状态，并在其满足条件时跳出循环，调用 early_put32 函数，使用 AUX_MU_IO_REG 寄存器实现对单个字符的输出。那么要实现输出一个字符串，我们只需要遍历此字符串，对其每个字符都调用一次 early_uart_send 函数即可。

```
79    /* LAB 1 TODO 3 BEGIN */
80    for (int i=0; str[i] != '\0'; ++i){
81        early_uart_send(str[i]);
82    }
83    /* LAB 1 TODO 3 END */
```

7 练习题

通过查阅 arm 的文档，可知 sctlr_el1 的第 0 位负责控制是否启用 MMU 的逻辑3。通过上下文代码可知，这里我们是使用通用寄存器 x8 来保存当前系统的配置信息。所以我们只需要使用按位或操作，把 sctlr_el1 中相应位的信息保存在 x8 中即可。而 arm 文档中也提供了启用 MMU 所对应的另一种方法3，即使用预定义的 #SCTLR_EL1_M 参数。实际上，下文的其他设定也都使用了这一方法。

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

AArch64 System Register Descriptions
D13.2 General system control registers

M, bit [0]

MMU enable for EL1&0 stage 1 address translation.

`0b0` EL1&0 stage 1 address translation disabled.
See the SCTLR_EL1.I field for the behavior of instruction accesses to Normal memory.

`0b1` EL1&0 stage 1 address translation enabled.

If the value of HCR_EL2.{DC, TGE} is not {0, 0} then in Non-secure state the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of the field.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to `0`.

图 3: sctlr_el1 寄存器说明