

Chcore Lab1 实验报告

519021910913 黄喆敏

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。

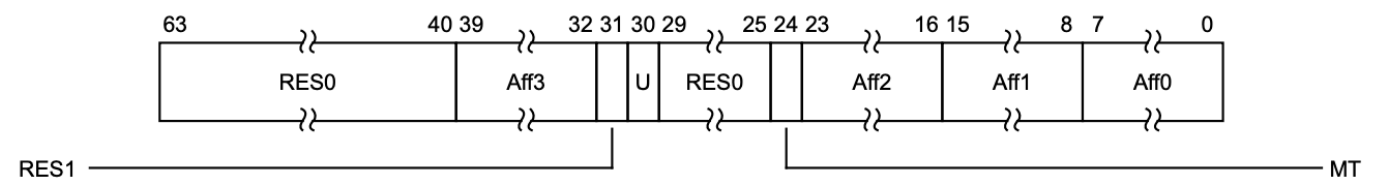
答：代码如下所示：

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

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

Field descriptions

The MPIDR_EL1 bit assignments are:



`mpidr_el1` 中的 `Aff0` 位存储了 core id。因此前两行将 `mpidr_el1` 的值存储到 `x8` 中，并保留低 8 位。若 `core id=0`，则进入 `primary` 函数，否则执行下一行 `b .`，即忙等。

即通过以上操作，将 `core id=0` 的核进入初始化流程，其他核暂停执行。

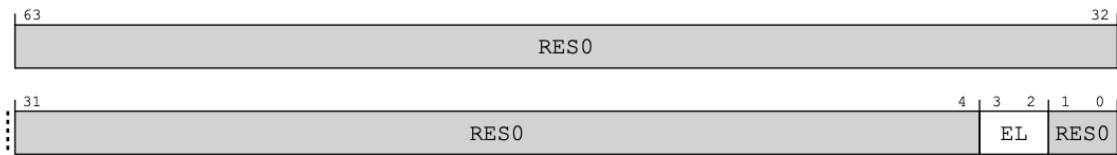
练习题 2：在 `arm64_elX_to_el1` 函数的 `LAB 1 TODO 1` 处填写一行汇编代码，获取 CPU 当前异常级别。

提示：通过 `CurrentEL` 系统寄存器可获得当前异常级别。通过 GDB 在指令级别单步调试可验证实现是否正确。

答：我们采用 `mrs x9, CurrentEL` 指令，并输出 `x9`，获取异常级别。

在 arm manual 的 C5-657 页，我们可以知道如何查看 `CurrentEL`。

Field descriptions



Bits [63:4]

Reserved, RES0.

EL, bits [3:2]

Current Exception level.

0b00 EL0.

0b01 EL1.

0b10 EL2.

0b11 EL3.

When the [HCR_EL2.NV](#) bit is 1, EL1 read accesses to the CurrentEL register return the value of 0b10 in this field.

The reset behavior of this field is:

- This field resets to the highest implemented Exception level.

Bits [1:0]

Reserved, RES0.

接着通过gdb调试，可知x9的值为0xc。因此当前异常级别为EL3。

```
os@ubuntu: ~/Desktop/chcore-lab
[ No Source Available ]

0x88000 <arm64_elX_to_el1> mrs x9, currentel
>0x88004 <arm64_elX_to_el1+4> cmp x9, #0x4
0x88008 <arm64_elX_to_el1+8> b.eq 0x88088 <arm64_elX_to_el1+136> //
0x8800c <arm64_elX_to_el1+12> cmp x9, #0x8
0x88010 <arm64_elX_to_el1+16> b.eq 0x88024 <arm64_elX_to_el1+36> //
0x88014 <arm64_elX_to_el1+20> mrs x9, scr_el3
0x88018 <arm64_elX_to_el1+24> mov x10, #0x501 //
0x8801c <arm64_elX_to_el1+28> orr x9, x9, x10
0x88020 <arm64_elX_to_el1+32> msr scr_el3, x9

remote Thread 1.1 In: arm64_elX_to_el1 L?? PC: 0x88004
(gdb) ni
0x000000000000088000 in arm64_elX_to_el1 ()
(gdb) i r x9
x9 0x0 0
(gdb) si
0x000000000000088004 in arm64_elX_to_el1 ()
(gdb) i r x9
x9 0xc 12
(gdb)
```

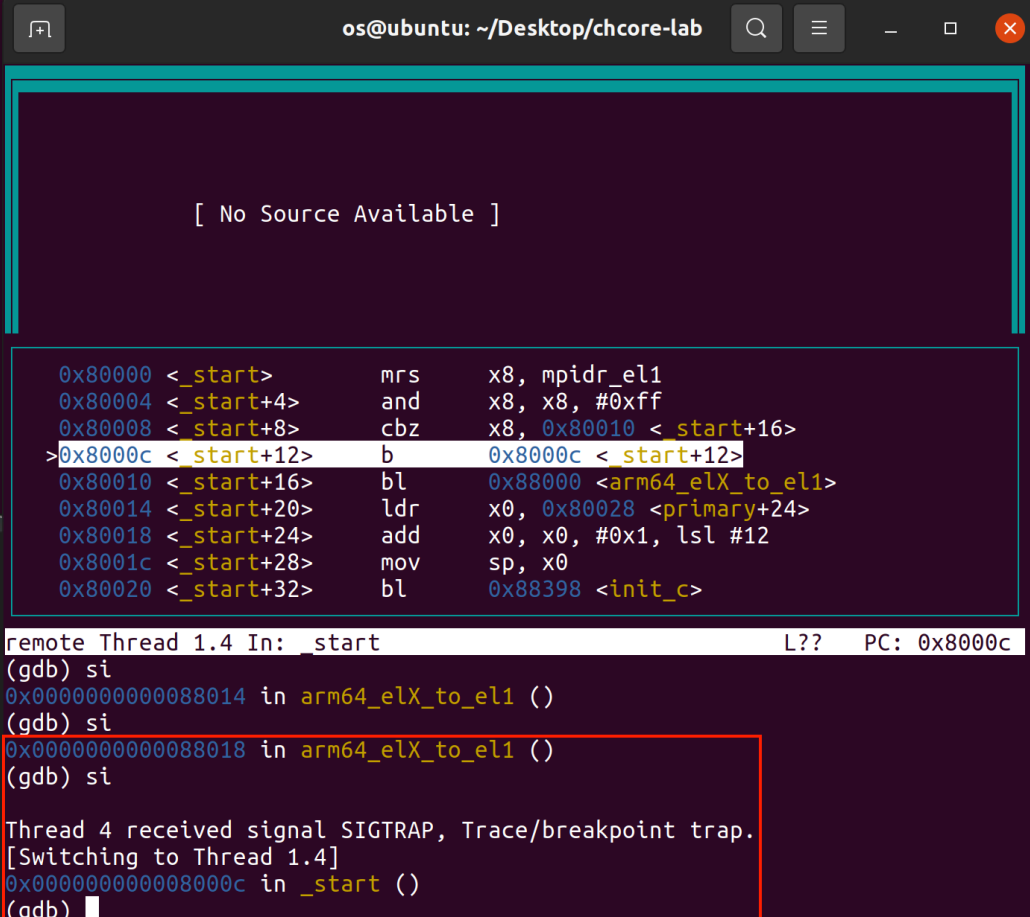
练习题 3: 在 `arm64_elx_to_el1` 函数的 LAB 1 TODO 2 处填写大约 4 行汇编代码, 设置从 EL3 跳转到 EL1 所需的 `elr_el3` 和 `spsr_el3` 寄存器值。具体地, 我们需要在跳转到 EL1 时暂时屏蔽所有中断、并使用内核栈 (`sp_el1` 寄存器指定的栈指针)。

答: 跳转到 EL1 时, `elr_el3` 中保存 `Ltarget` 所在地址, 且 `spsr_el3` 中需要保存 EL3 的处理器状态, 即 `SPSR_ELX_DAIIF | SPSR_ELX_EL1H`。

因此代码为:

```
adr x9, .Ltarget
msr elr_el3, x9
mov x9, SPSR_ELX_DAIIF | SPSR_ELX_EL1H
msr spsr_el3, x9
```

我们采用 gdb 单步调试, `arm64_elx_to_el1` 可以正常返回到 `_start`, 因此代码正确。



The screenshot shows a GDB terminal window titled "os@ubuntu: ~/Desktop/chcore-lab". The main display area shows "[No Source Available]". Below this, a list of assembly instructions is displayed, with the instruction at address 0x8000c highlighted: `>0x8000c <_start+12> b 0x8000c <_start+12>`. The instructions include `mrs x8, mpidr_el1`, `and x8, x8, #0xff`, `cbz x8, 0x80010 <_start+16>`, `bl 0x88000 <arm64_elx_to_el1>`, `ldr x0, 0x80028 <primary+24>`, `add x0, x0, #0x1, lsl #12`, `mov sp, x0`, and `bl 0x88398 <init_c>`. Below the assembly list, the GDB prompt shows the execution of `remote Thread 1.4 In: _start`, `(gdb) si`, and `0x00000000000088014 in arm64_elx_to_el1 ()`. The next `(gdb) si` command leads to `0x00000000000088018 in arm64_elx_to_el1 ()`. A message indicates "Thread 4 received signal SIGTRAP, Trace/breakpoint trap." and the prompt switches to Thread 1.4, showing `0x0000000000008000c in _start ()` and `(gdb) |`.

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?

答: 因为 C 函数需要利用栈, 进行传递参数, 保存/恢复寄存器, 保存返回地址等工作。因此首先要设置足够大小的启动栈, 使程序正常执行; 若不设置, 上述工作无法执行, 可能会出现栈溢出, 程序崩溃等问题。

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。

答: `.bss` 段包含了未初始化的全局变量和静态变量, 其初始值是未知的, 因此必须要清零。若不清理 `.bss` 段, 在运行后续代码时变量的值可能不为 0, 可能会导致一些潜在的错误, 内核无法工作。

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。

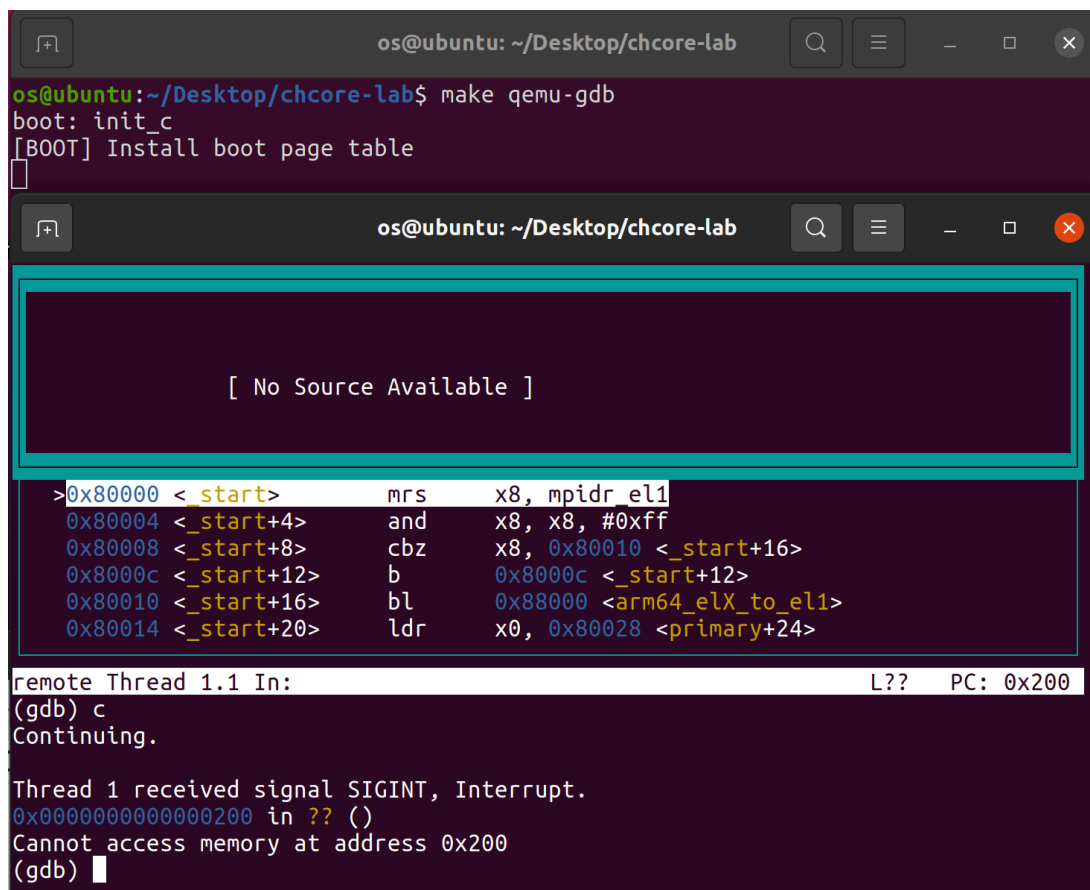
答: 将每个字符依次输出即可。

```
void uart_send_string(char *str)
{
    for(int i = 0; str[i] != '\0'; i++) {
        early_uart_send((unsigned int) str[i]);
    }
}
```

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。

答: SCTLR_EL1 寄存器的 M 字段用来启用/禁用 MMU。因此我们采用 `orr x8, x8, #SCTLR_EL1_M` 指令, 将 M 字段设为 1, 启用 MMU。

我们在 QEMU 中验证。可以发现执行流在 `0x200` 处无限循环, 符合要求。



```
os@ubuntu: ~/Desktop/chcore-lab
os@ubuntu:~/Desktop/chcore-lab$ make qemu-gdb
boot: init_c
[BOOT] Install boot page table
[ No Source Available ]
>0x80000 <_start>      mrs      x8, mpidr_el1
0x80004 <_start+4>     and       x8, x8, #0xff
0x80008 <_start+8>     cbz       x8, 0x80010 <_start+16>
0x8000c <_start+12>    b         0x8000c <_start+12>
0x80010 <_start+16>    bl        0x88000 <arm64_elX_to_el1>
0x80014 <_start+20>    ldr       x0, 0x80028 <primary+24>
remote Thread 1.1 In: L?? PC: 0x200
(gdb) c
Continuing.
Thread 1 received signal SIGINT, Interrupt.
0x0000000000000200 in ?? ()
Cannot access memory at address 0x200
(gdb)
```