Chcore Lab1

520030910393 马逸川

首先感谢助教学长的指导。

思考题1:

Refrence:https://developer.arm.com/

_start函数开头部分如下:

```
 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>
```

其中，`mpidr_el1`寄存器对每个cpu核不同，在多处理器的系统中为调度提供标识。

在`mpidr_el1`的各位中，Aff0(bits[7:0])用于区分各个核心，thread1.1的标识为0，其余均不为0。基于此，在`0x80008`判断最后一位是否为0，若是则进入初始化，否则陷入死循环产生异常中断，暂停执行。

练习题2:

添加代码如下:

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

由下方几行的代码得知取得的CurrentEL放在x9寄存器中。

```
cmp x9, CURRENTEL_EL1
beq .Ltarget
cmp x9, CURRENTEL_EL2
beq .Lin_el2
```

练习题3:

添加代码如下:

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

这段代码的实现参考了课件和`.Lin_el2`代码段下的内容。

具体完成的思路是：

前两行:将`.Ltarget`位置地址取到x9寄存器中，作为el3的异常状态返回地址，存储于`elr_el3`中。

后两行:将el1状态的程序状态存储到x9中，再存储于`spsr_el3`，从而修改返回后的程序状态。

思考题4:

在进入`init_c.c`函数之前，通过以下的代码段设置栈。

```
   0x80014 <_start+20>     ldr     x0, 0x80028 <primary+24>
   0x80018 <_start+24>     add     x0, x0, #0x1, lsl #12
  >0x8001c <_start+28>     mov     sp, x0
```

参考`kernel.img`的反汇编。

```
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>
```

在进入`_init.c`之前，为`init_c.c`中的栈boot_cpu_stack提供足够大小的栈空间。

如果不设置，则随程序执行，sp会和存储在这段栈空间的数据发生冲突。

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。

参考附录，查看.bss段信息:

```
[ 6] .bss              NOBITS           ffffff00000b0040  0004003f
     0000000000008000  0000000000000000  WA       0     0     16
```

如果未清零.bss段，内核在使用全局变量时可能调用随机的初始值，可能会为变量分配错误的值和大小。

练习题6:

阅读`uart.c`中实现的函数，在TODO处填写如下代码:

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

初始化UART，取每个字符并输出，结果如下:

```
os@ubuntu:~/Desktop/chcore-lab$ make qemu
boot: init_c
[BOOT] Install boot page table
```

练习题7:

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

这一段参考下方的几行实现的。可以看到调用`SCTLR_EL1`寄存器各个字段的方式，参考得出调用M字段的代码。

```
bic     x8, x8, #SCTLR_EL1_A
bic     x8, x8, #SCTLR_EL1_SA0
bic     x8, x8, #SCTLR_EL1_SA
orr     x8, x8, #SCTLR_EL1_nAA
```