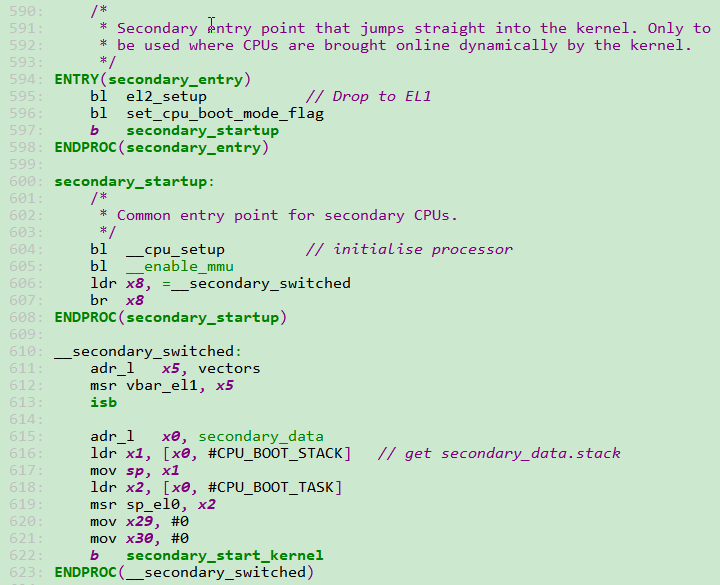
文件：arch/arm64/kernel/head.S

其他core的入口：



**secondary\_entry：**

在从bl31切到EL1上的Linux Kernel后：

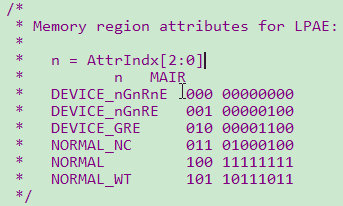
第595行，在el2\_setup中设置EL1和EL0为小端模式，然后将w0设置为BOOT\_CPU\_MODE\_EL1，并返回

第596行，记录boot cpu的启动模式到\_\_boot\_cpu\_mode，目前是BOOT\_CPU\_MODE\_EL1

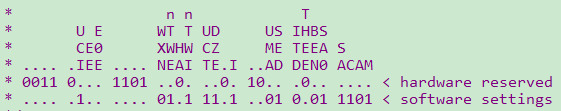
**secondary\_startup：**

第604行，\_\_cpu\_setup:

1. 无效本地tlb
2. 使能FP/ASIMD
3. 设置mdscr\_el1，在EL0访问Debug Communication Channel寄存器时，陷入EL1
4. 操作daif，使能debug中断
5. 设置pmuserenr\_el0，当EL0访问PMU寄存器时会陷入EL1
6. 填充mair\_el1，设置后面要用到的内存属性索引，目前用到了6中内存属性：



1. 读取sctlr\_el1，修改后存入x0，后面配置mmu时会用到x0的值：x0= (sctrl\_el1 & ~0xfcffffff)|0x34d5d91d，即将控制大小端的bit保留（因为之前在el2\_setup里设置过了），其他位清零，然后设置新值，新值的含义如下：



从低位到高位依次说明:

|  |  |  |
| --- | --- | --- |
| 0 | M | 1：表示开启EL1和EL0的stage1地址转换机制，目前因为用不到虚拟化，所有只有stage1 |
| 1 | A | 0：关闭地址对齐检查，但是load/store exclusive和load-acquire/store-release除外 |
| 2 | C | 1：EL0/1的data cache的控制不再由sctrl\_el1控制，如果HCR\_EL2.DC为1，那么EL0/1的data cache开启。（所以，不开MMU，数据cache也可以开？） |
| 3 | SA | 1 EL1上栈指针对齐检查，需要16字节对齐 |
| 4 | SA0 | 1：EL0上栈指针对齐检查，需要16字节对齐 |
| 5 | CP15BEN | 0：EL0运行在AArch32时，不能使用CP15DMB、CP15DSB以及CP15ISB |
| 7 | ITD | 0：EL0运行在AArch32模式时，仍旧可以使用IT指令（IF-THEN）<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489c/Cjabicci.html> |
| 8 | SED | 1：EL0运行在AArch32时，不能使用SEDEND指令。这个指令是在ARMv6引入的，用于切换当前cpu的大小端，请参考：<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489c/Cjacabbf.html> |
| 9 | UMA | 0：在EL0运行在AArch64上通过MRS或MSR指令访问DAIF时会陷入EL1 |
| 12 | I | 1：EL0/1的指令cache不再由sctrl\_el1控制。如果HCR\_EL2.DC为1，那么EL0/1的指令 cache开启 |
| 14 | DZE | 1：EL0运行AArch64时，如果执行DC ZVA指令（清除指令虚拟地址的data cache），会陷入EL1. |
| 15 | UCT | 1：EL0运行AArch64时，访问CTR\_EL0时不会陷入EL1. 这个寄存器用于获取当前系统的cache的架构信息 |
| 16 | nTWI | 1: EL0上执行WFI时不会陷入EL1 |
| 18 | nTWE | 1: EL0上执行WFE时不会陷入EL1 |
| 19 | WXN | 0: 对EL1&0的内存访问权限没有影响。如果是1的话，在EL1&0下，如果某块内存具备可写权限，那么这块内存就是XN（Execute Never） |
| 24 | E0E | 控制EL0数据访问的大小端控制，0小端，1大端 |
| 25 | EE | 控制EL1以及EL1/0的table walk时的大小端，0小端，1大端 |
| 26 | UCI | 1：EL0运行在AArch64时，如果执行DC CVAU/CIVAC/CVAC、IC IVAU时不会陷入EL1 |

M:

8、