指令 la sp, bootstacktop

la（load address）用于将标签 bootstacktop 的地址加载到寄存器 sp（stack pointer，栈指针寄存器）

- bootstacktop 是在 .data 段中定义的标签，位于内核启动栈（bootstack）的顶部。
- 该指令将 bootstacktop 的物理地址加载到 sp 寄存器中，从而初始化栈指针。

目的：确保内核在后续执行中有有效的栈空间，避免了因栈未初始化导致的崩溃或未定义行为。

指令 tail kern_init

tail kern_init 是一条 RISC-V 尾调用指令，用于跳转到标签 kern_init 的地址。

- tail 指令执行一个跳转操作，类似于 j（jump），但它优化了函数调用的开销。它不会将返回地址保存到链接寄存器（ra，即 x1），也不会修改当前栈帧，而是直接复用当前函数的上下文。
- 在这里，tail kern_init 从 kern_entry 跳转到 kern_init 函数，并将控制权完全转移给 kern_init，且 kern_entry 不会期望返回（即 kern_entry 的栈帧被丢弃）。

目的：高效地进入内核的主初始化阶段。

RISC-V 硬件加电后最初执行的几条指令位于 0x80000000 地址。

功能：

1. 硬件初始化

- **设置关键寄存器：初始化栈指针(sp)、帧指针(fp)等基础寄存器**
- **内存控制器初始化：配置 DDR 内存控制器，确保内存可访问**
- **时钟和电源管理：设置系统时钟频率和电源管理单元**

2. 引导加载程序执行

- **设备探测：检测可用的存储设备**
- **内核加载：从存储设备将操作系统内核映像加载到内存中**

- 校验完整性：验证内核映像的完整性和有效性

3. 环境准备
   - 栈空间设置：为 C 代码执行准备栈空间（bootstack）
   - 基本异常处理：设置初步的异常处理例程
   - 硬件模式设置：配置 CPU 的工作模式
4. 跳转到内核入口
   - 执行流程跳转到 kern_entry（地址 0x80200000）
   - 此时栈指针(sp)已初始化为 0x8001bd80，表明基础环境已准备就绪

（这里 ai 写的有几点我按照他给的调试过程没看出来，要是你们觉得哪不对劲就别往上加了）

调试过程：

1. 设置断点
   对 kern_entry 函数下断点
   (gdb) b* kern_entry
   随后执行 continue(缩写为 c)命令

```
(gdb) c
Continuing.

Breakpoint 1, kern_entry () at kern/init/entry.S:7
7           la sp, bootstacktop
```

2. 使用 info registers（可简写为 i r）查看所有寄存器的值

```
(gdb) i r
ra             0x80000a02       0x80000a02
sp             0x8001bd80       0x8001bd80
gp             0x0      0x0
tp             0x8001be00       0x8001be00
t0             0x80200000       2149580800
t1             0x1      1
t2             0x1      1
fp             0x8001bd90       0x8001bd90
s1             0x8001be00       2147597824
a0             0x0      0
a1             0x82200000       2183135232
a2             0x80200000       2149580800
a3             0x1      1
a4             0x800    2048
a5             0x1      1
a6             0x82200000       2183135232
a7             0x80200000       2149580800
s2             0x800095c0       2147521984
s3             0x0      0
s4             0x0      0
s5             0x0      0
s6             0x0      0
s7             0x8      8
s8             0x2000   8192
s9             0x0      0
s10            0x0      0
s11            0x0      0
t3             0x0      0
t4             0x0      0
t5             0x0      0
t6             0x82200000       2183135232
pc             0x80200000       0x80200000 <kern_entry>
dscratch       Could not fetch register "dscratch"; remote failure reply 'E14'
mucounteren    Could not fetch register "mucounteren"; remote failure reply 'E14'
```

3. 详解

ra(Return Address) - 存放函数调用后的返回地址

sp(Stack Pointer) - 指向当前栈的顶部

gp(Global Pointer) - 一个优化用的寄存器，通常指向全局数据区的一个中间位置，以便用更短的指令访问更多的全局变量。

t0- t2, a0- a7-临时寄存器和用于传递函数参数的寄存器

s0/ fp(Frame Pointer)- 帧指针，用于定位当前函数的栈帧，便于调试和回溯。

s1- s11-在函数调用中需要被被调用者保存的寄存器。

pc(Program Counter) -当前正在执行的指令的地址。