练习一：理解内核启动中的程序入口操作

entry.S文件是引导加载程序运行时跳转到的文件，里面包含内核栈空间的分配和内核初始化。

1. la sp, bootstacktop 指令是load address的伪指令，sp栈指针寄存器指向bootstack的顶部也就是内核栈的高地址起始位置。汇编器会把它编译成：

   ```
   auipc sp, %pcrel_hi(bootstacktop)
   addi  sp, sp, %pcrel_lo(bootstacktop)
   ```

   目的：为内核准备安全的栈空间，存储函数调用的返回地址，局部变量等。

2. tail kern_init指令是跳转到c程序kern_init，执行内核初始化流程。**目的**：启动内核初始化，建立页表；初始化内存管理；设置中断；启动调度；进入主循环。

练习二：使用GDB验证启动流程

CPU加电复位后从固定复位地址开始执行也就是ox1000
最初几条指令位于 **0x1000**（OpenSBI 固件入口），简要功能如下：

1. **设置堆栈指针（SP）** → 初始化 CPU 栈
2. **设置全局指针（GP）** → 便于访问全局数据
3. **初始化内存和页表** → 为内核加载做好准备
4. **初始化基本设备** → 如 UART 控制台
5. **跳转到内核入口** → 把控制权交给内核（地址 0x80200000）

如图，**PC** 已经指向内核入口第一条指令。

此时还没执行第一条指令

接下来单步执行指令si

观察堆栈初始化并跟踪 OpenSBI 到内核加载过程：

watch：监控内核写入内存的瞬间，CPU 会在内核写入 `0x80200000` 时暂停，用 `info registers pc` 查看 PC，确认内核已经被加载。

下一步继续单步跟踪，观察堆栈指针、全局指针、函数参数等寄存器的初始化

以上是完整调试过程，回答问题：**RISC-V 硬件加电后最初执行的几条指令位于什么地址？它们主要完成了哪些功能？**

RISC-V 硬件加电后的最初执行地址

- **复位地址**：RISC-V CPU 加电后最先执行的指令位于 `0x1000`。
- 这些地址对应的是 **SBI 固件（OpenSBI）的汇编代码段。**

最初几条指令的主要功能：

1. 0x1000：auipc t0,0x0

   生成 PC 相对地址，初始化 t0，为后续内存访问或跳转做准备。

   初始化堆栈指针为固件本身的运行设置临时堆栈

2. 0x1004：addi a1,t0,32 #0x1020

   t0 + 32 → a1，用作下一步跳转或参数传递，准备进入 OpenSBI 主入口。

3. 0x1008 csrr a0,mhartid 读取当前 hart（CPU 核心）ID，存入 a0，多核初始化需要。

4. 0x100c ld t0,24(t0) 从 t0+24 的内存地址加载值到 t0，通常是读取跳转地址或固件参数。

```
0x1010-0x80000000
```

进入 OpenSBI 的初始化函数，主要完成 **最基础的硬件初始化**：

- 配置 CPU 寄存器（CSR）
- 设置中断控制器
- 初始化内存布局（如清零 BSS 段）
- 为内核加载做准备

总结：本实验中重要的知识点以及对应得os原理中的知识点（待队友补充）

1. RISCV加点复位和启动流程：实验通过qemu+GDb的模拟与调试方式，模拟了cpu从加电到将进入entry的初始化的过程中的一系列变化，包括sp指针的指向以及从汇编转为c语言。**差异**：实验中更低层，涉及具体硬件地址和指令；OS原理课程中更抽象，只讲 CPU 启动流程和 OS 初始化概念。
2. PC、SP、寄存器状态观察：通过info register指令观察各个寄存器状态以及栈的使用情况，偏于理解函数调用与寄存器使用。**差异**：实验可以看到真实地址和寄存器变化；原理课程更抽象，注重概念。
3. 内核映像加载与地址空间布局：通过 watch *0x80200000 观察内核被写入；这对应 OS 原理中“从磁盘加载内核到内存后执行”的阶段。差异在于实验中由 OpenSBI 完成内存拷贝，而原理课中通常用引导加载器描述这个过程。
4. 控制权转移：实验通过 `b *0x80200000` 验证了控制权的移交，这是理论中的“从引导程序进入操作系统”的关键一步。差异在于理论上只讲“加载并跳转”，实验中可以在 GDB 中精确看到这条跳转指令的执行。

本实验未涉及(课堂见过)：

1. 进程调度与上下文切换，OS原理课程重点讲进程状态、调度算法、寄存器保存与恢复。
2. 虚拟内存与分页机制，内核管理虚拟地址空间，页表转换机制。
3. 文件系统与 I/O 子系统：内核管理磁盘数据和目录结构

**以下是运行到断点处再往后逐步运行几条指令，观察后继变化

现在已经：

1. 用 QEMU 模拟了 RISC-V CPU
2. 用 GDB 在内核入口停下
3. 看到了内核 **第一条汇编指令**在做栈初始化
4. 下一步就是单步跟踪 `kern_init` 的执行

在向下运行一条指令查看寄存器状态，发现bootstacktop的地址传给sp指针。

接下来执行kern_inti，pc寄存器地址发生改变，说明内核启动流程已经从 **汇编入口（`kern_entry`）转到 C 语言初始化（`kern_init`）**