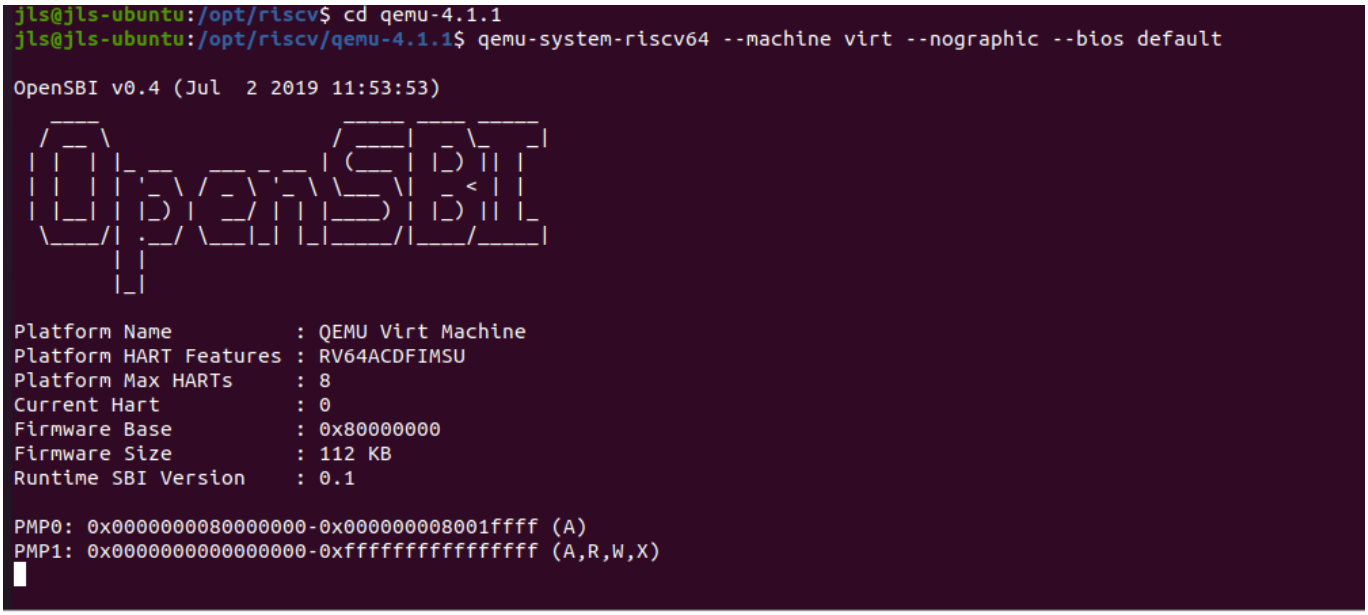
lab1

最小可执行内核：在该实验中我们建立一个最小的可执行内核，它能够进行格式化的输出并进入死循环。

我采用了Ubuntu20.04虚拟机，搭配qemu4.1.1，结合riscv交叉编译链的环境配置完成实验，环境配置完成后通过如下形式进行了验证



练习1：理解内核启动中的程序入口操作

阅读 kern/init/entry.S内容代码，结合操作系统内核启动流程，说明指令 la sp, bootstacktop 完成了什么操作，目的是什么？ tail kern_init 完成了什么操作，目的是什么？

首先我们先观察一下entry.s的代码我们可以观察到代码段，数据段定义，内核入口点声明，栈指针初始化等一系列操作

```
#include <mmu.h>
#include <memlayout.h>

    .section .text,"ax",%progbits
    .globl kern_entry
kern_entry:
    la sp, bootstacktop

    tail kern_init

.section .data
    # .align 2^12
    .align PGSHIFT
    .global bootstack
bootstack:
    .space KSTACKSIZE
    .global bootstacktop
bootstacktop:
```

la sp, bootstacktop 是一条伪指令，将 bootstacktop 的地址加载到栈指针寄存器 sp 中。

其目的是初始化内核的栈指针，为后续C代码的执行准备栈空间。bootstacktop 指向预先分配的内核栈的顶部，栈从高地址向低地址增长

tail kern_init 是一条尾调用指令，跳转到 kern_init 函数，并且不保留返回地址。

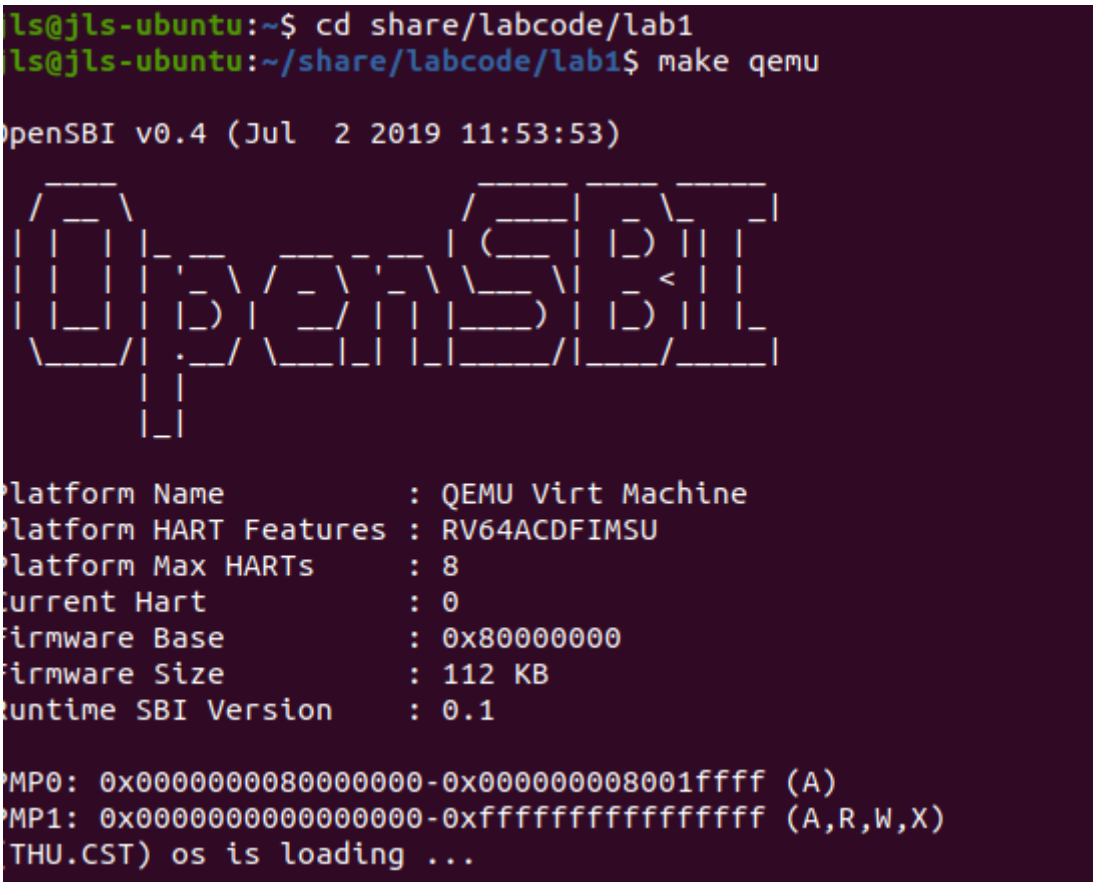
其目的是跳转到内核的C语言初始化函数，开始执行内核的主要初始化工作。使用尾调用可以节省栈空间，因为不需要保存返回地址

练习2：使用GDB验证启动流程

为了熟悉使用 QEMU 和 GDB 的调试方法，请使用 GDB 跟踪 QEMU 模拟的 RISC-V 从加电开始，直到执行内核第一条指令（跳转到 0x80200000）的整个过程。通过调试，请思考并回答：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？请在报告中简要记录你的调试过程、观察结果和问题的答案。

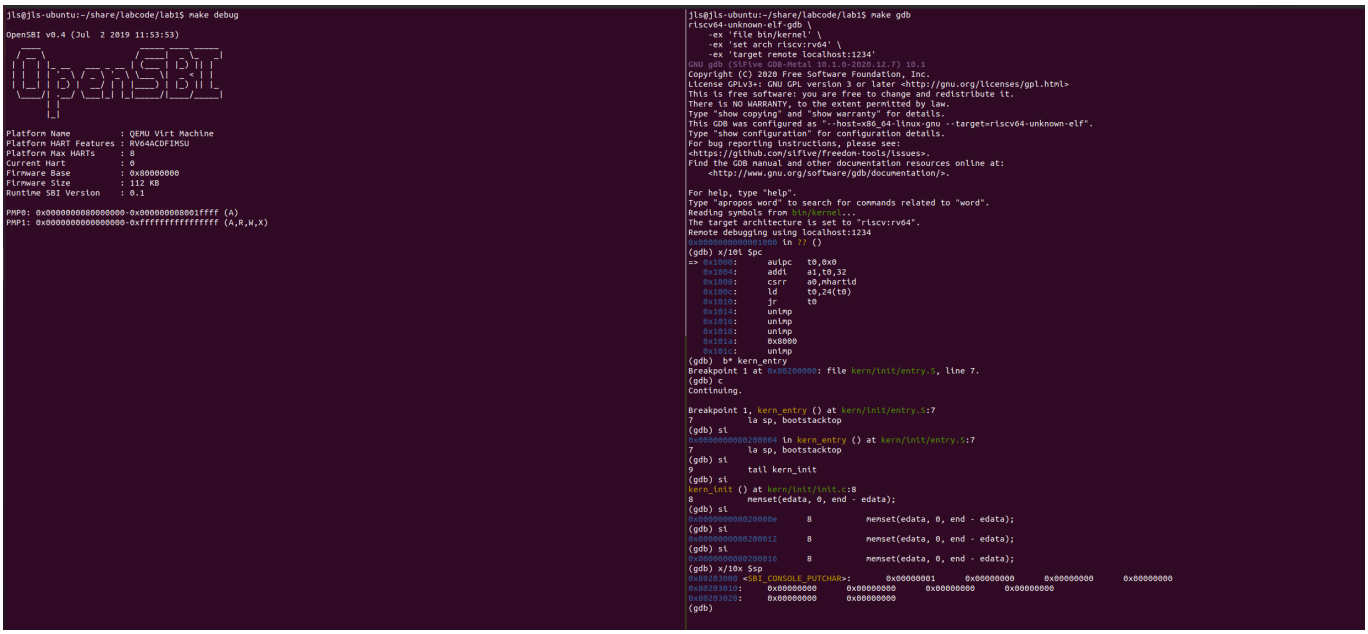
实验过程：

首先我们先来叙述一下相关的实验过程，这里我们先进入到实验代码的目录下 通过make qemu 指令运行代码，确保我们的环境以及代码都没有错误。成功运行如下



测试完成后我们进行实验，先通过tmux指令分出两个端口，我们在左侧端口输入make debug指令启动qemu,右侧端口输入make gdb 连接调试器。

输入相关指令后得到的输出结果如下：



通过观察我们发现初始状态下程序计数器$pc位于0x1000,沃我们使用x/10i pc来查看最初的几条指令

```
0x1000:      auipc   t0,0x0          将当前PC值加上偏移量存入t0寄存器
0x1004:      addi    a1,t0,32        计算地址偏移
0x1008:      csrr    a0,mhartid      读取当前硬件线程ID
0x100c:      ld      t0,24(t0)       从内存加载数据到t0寄存器
0x1010:      jr      t0              跳转到目标地址
0x1014:      unimp
0x1016:      unimp
0x1018:      unimp
0x101a:      0x8000
0x101c:      unimp
```

这些指令完成了早期的硬件初始化和引导加载程序（OpenSBI）的启动准备

接着我们在内核入口处设置断点(b* kern_entry),使用 c 继续执行，在 0x80200000 处触发断点,此时进入 kern/init/entry.S 的 kern_entry 函数

接着我们使用 si 单步执行，观察栈指针初始化和跳转到 kern_init 的过程，最终进入 kern/init/init.c 的 kern_init 函数。

有调试结果我们得出结论RISC-V 硬件加电后最初执行的几条指令位于0x1000到0x1010之间，从 0x1000 到 0x1010 包含了完整的初始指令序列，这个指令序列完成了从加电到跳转到下一阶段（OpenSBI）的完整过程。

重要知识点对应

实验知识点	OS原理知识点	实验含义	原理含义	关系与差异
栈指针初始化	进程/线程栈管理	为内核设置初始栈空间	操作系统为每个进程分配栈空间	实验中是内核初始化时的一次性栈设置，原理中是动态的进程栈管理

实验知识点	OS原理知识点	实验含义	原理含义	关系与差异
内核入口点	系统启动流程	内核代码的起始执行点	操作系统从BIOS/固件到内核的过渡	实验展示了具体的跳转过程，原理描述抽象流程
内存布局	内存管理	通过链接脚本定义的内核内存分布	操作系统的虚拟内存管理	实验是静态布局，原理包含动态分配和管理
固件交互	系统引导	OpenSBI提供基本服务	BIOS/UEFI等引导程序	RISC-V使用OpenSBI，与传统x86架构不同

重要但未涉及的知识点我认为有 进程调度，虚拟内存，并发控制等