练习1：使用GDB验证启动流程

在 lab0 目录下，使用 `make debug` 和 `make gdb` 指令进入调试。使用 gdb 调试 QEMU 模拟的 RISC-V 计算机加电开始运行到执行应用程序的第一条指令（即跳转到0x80200000）这个阶段的执行过程。说明RISC-V硬件加电后的几条指令在哪里？完成了哪些功能？

1. 调试过程

- 使用 `x/10i $pc` 指令显示即将执行的10条汇编指令。

```
0x0000000000001000 in ?? ()
(gdb) x/10i $pc
=> 0x1000:      auipc   t0,0x0
   0x1004:      addi    a1,t0,32
   0x1008:      csrr    a0,mhartid
   0x100c:      ld      t0,24(t0)
   0x1010:      jr      t0
   0x1014:      unimp
   0x1016:      unimp
   0x1018:      unimp
   0x101a:      0x8000
   0x101c:      unimp
```

- 使用 `info r t0` 指令显示 t0 寄存器的值。
- 使用 `si` 指令单步执行一条汇编指令。

```
(gdb) info r t0
t0             0x0      0
(gdb) si
0x0000000000001004 in ?? ()
(gdb) info r t0
t0             0x1000   4096
(gdb) si
0x0000000000001008 in ?? ()
(gdb) si
0x000000000000100c in ?? ()
(gdb) si
0x0000000000001010 in ?? ()
(gdb) info r t0
```

```
t0             0x80000000       2147483648
(gdb) si
0x0000000080000000 in ?? ()
```

- 使用 `x/10i $pc` 指令显示即将执行的10条汇编指令。

```
0x0000000080000000 in ?? ()
(gdb) x/10i $pc
=> 0x80000000:  csrr    a6,mhartid
   0x80000004:  bgtz    a6,0x80000108
   0x80000008:  auipc   t0,0x0
   0x8000000c:  addi    t0,t0,1032
   0x80000010:  auipc   t1,0x0
   0x80000014:  addi    t1,t1,-16
   0x80000018:  sd      t1,0(t0)
   0x8000001c:  auipc   t0,0x0
   0x80000020:  addi    t0,t0,1020
   0x80000024:  ld      t0,0(t0)
```

- 使用 `break *0x80200000` 指令在 0x80200000 处设置断点。
- 使用 `continue` 指令执行直到碰到断点。

```
(gdb) break *0x80200000
Breakpoint 1 at 0x80200000: file kern/init/entry.S, line 7.
(gdb) continue
Continuing.

Breakpoint 1, kern_entry () at kern/init/entry.S:7
7           la sp, bootstacktop             # 加载 bootstacktop 地址到栈指针 (sp) 寄
存器，初始化栈顶。
```

2. 回答问题

RISC-V 硬件加电后的几条指令在以物理地址 0x1000 开头的区域上。之后跳转到以物理地址 0x80000000 开头的区域上，再之后跳转到以物理地址 0x80200000 开头的区域上。

完成的功能如下：

a. 复位代码 (位于物理地址 0x1000)

- 0x1000: auipc t0,0x0：这是一条立即数偏移指令，它将 PC 的高 20 位与 0 相加，并将结果存储在 t0 寄存器中。由于 PC 当前指向 0x1000，这条指令实际上设置了 t0 为 0x1000。
- 0x1004: addi a1,t0,32：将 t0 (0x1000) 加上 32，结果存储在 a1 寄存器中。这可能是指向一个数据结构或配置表的地址。
- 0x1008: csrr a0,mhartid：从特权模式下的 mhartid 寄存器读取当前硬件线程的 ID 并存储在 a0 寄存器中。mhartid 通常用于多核系统中标识不同的 CPU 核心。

- 0x100c: ld t0,24(t0)：从 t0 + 24 (即 0x1024) 的地址加载一个 64 位的数据到 t0 寄存器中。这个地址可能存储了下一个要执行的代码段的入口地址。
- 0x1010: jr t0：跳转到 t0 寄存器中的地址继续执行。在这个例子中，t0 被设置为 0x80000000，所以控制流跳转到了这个地址。

b. Bootloader代码 (位于物理地址 0x80000000)

- 0x80000000: csrr a6,mhartid：再次读取当前硬件线程的 ID，这次是为了检查是否是主核心。
- 0x80000004: bgtz a6,0x80000108：如果 a6 (mhartid) 不等于 0，则跳转到 0x80000108。如果不是主核心（mhartid = 0），则跳过一些初始化代码。
- 0x80000008 至 0x80000024：这部分代码准备一些数据结构，例如初始化堆栈指针等。

c. 跳转到操作系统内核入口 (0x80200000)

- 0x80200000: la sp, bootstacktop：设置栈指针 (sp) 到 bootstacktop，这是初始化堆栈的一个重要步骤。bootstacktop 是预先定义好的一个符号，代表了堆栈的顶端地址。

3. 练习总结

- RISC-V 硬件加电后首先执行的是位于 0x1000 的复位代码，该代码负责基本的初始化工作，如读取硬件线程 ID 和加载下一阶段引导代码的地址。
- 然后，通过无条件跳转指令转移到 0x80000000 处的Bootloader代码，这里进行了更详细的初始化，包括多核环境下的核心识别和资源分配。
- 最后，引导过程会跳转到 0x80200000，这是操作系统内核的入口点，在这里设置了正确的堆栈并开始执行主要的程序逻辑。

在QEMU模拟的这款riscv处理器中，将复位向量地址初始化为0x1000，再将PC初始化为该复位地址，因此处理器将从此处开始执行复位代码，复位代码主要是将计算机系统的各个组件（包括处理器、内存、设备等）置于初始状态，并且会启动Bootloader，在这里QEMU的复位代码指定加载Bootloader的位置为0x80000000，Bootloader将加载操作系统内核并启动操作系统的执行。