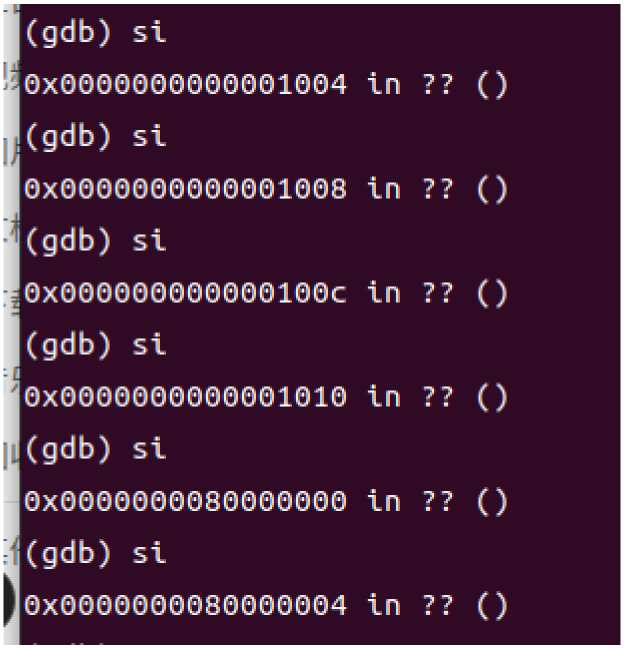
使用GDB验证启动流程

相关代码

```
(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
```



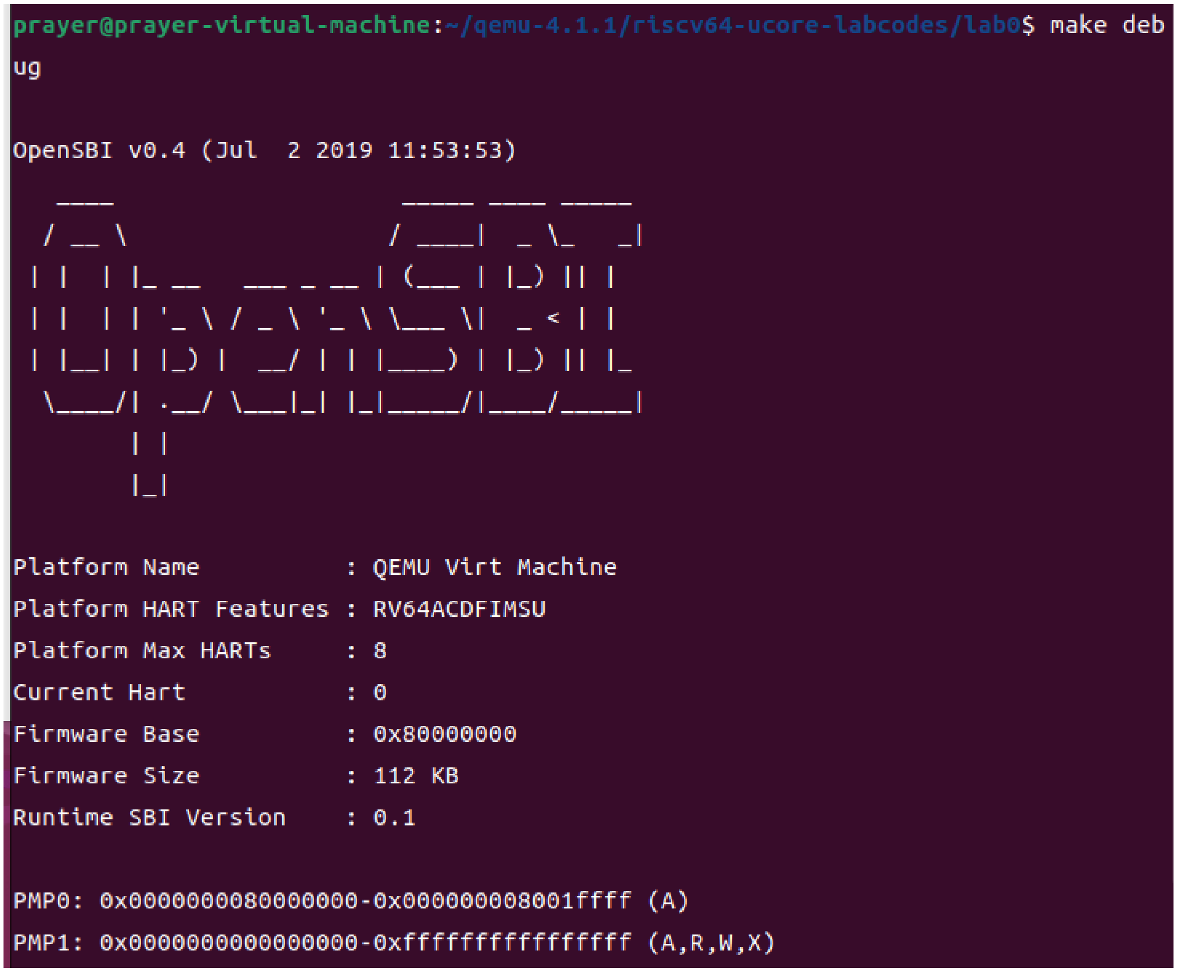
```
(gdb) si
0x0000000000001004 in ?? ()
(gdb) si
0x0000000000001008 in ?? ()
(gdb) si
0x000000000000100c in ?? ()
(gdb) si
0x0000000000001010 in ?? ()
(gdb) si
0x0000000080000000 in ?? ()
(gdb) si
0x0000000080000004 in ?? ()
```

经过si单步执行汇编指令，发现执行完0x1010: jr t0后，pc跳转至0x80000000。

即开始跳转至bootloader，开始运行bootloader，负责开机并加载操作系统至内存中。

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

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



```
prayer@prayer-virtual-machine:~/qemu-4.1.1/riscv64-ucore-labcodes/lab0$ make deb
ug

OpenSBI v0.4 (Jul  2 2019 11:53:53)
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name          : QEMU Virt Machine
Platform HART Features : RV64ACDFIMSU
Platform Max HARTs     : 8
Current Hart           : 0
Firmware Base          : 0x80000000
Firmware Size          : 112 KB
Runtime SBI Version    : 0.1

PMP0: 0x0000000080000000-0x000000008001ffff (A)
PMP1: 0x0000000000000000-0xffffffffffffffff (A,R,W,X)
```

可以发现当运行至0x80200000处时，内核镜像os.bin开始被加载运行，操作系统得到启动。

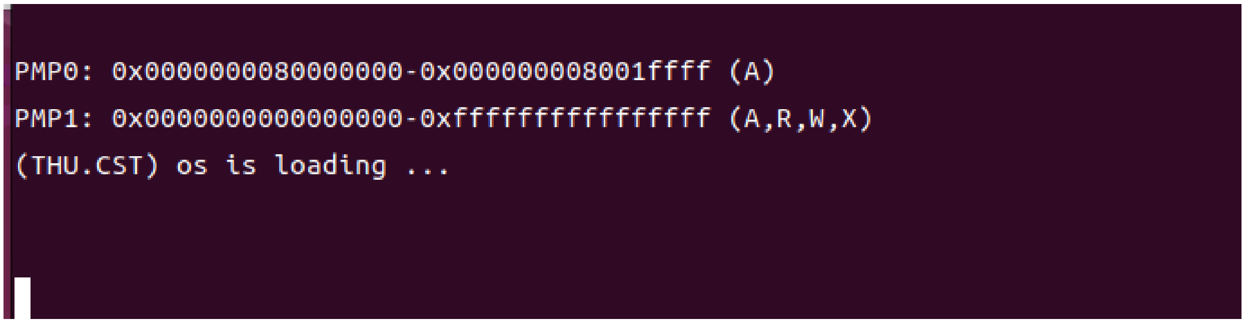
```
(gdb) b *0x80200000
Breakpoint 1 at 0x80200000: file kern/init/entry.S, line 7.
(gdb) c
Continuing.

Breakpoint 1, kern_entry () at kern/init/entry.S:7
7           la sp, bootstacktop
(gdb) x/100i $pc
=> 0x80200000 <kern_entry>: auipc   sp,0x3
   0x80200004 <kern_entry+4>:   mv   sp,sp
   0x80200008 <kern_entry+8>:   j    0x8020000c <kern_init>
   0x8020000c <kern_init>:  auipc   a0,0x3
   0x80200010 <kern_init+4>:    addi    a0,a0,-4
   0x80200014 <kern_init+8>:    auipc   a2,0x3
   0x80200018 <kern_init+12>:   addi    a2,a2,-12
   0x8020001c <kern_init+16>:   addi    sp,sp,-16
   0x8020001e <kern_init+18>:   li   a1,0
   0x80200020 <kern_init+20>:   sub a2,a2,a0
   0x80200022 <kern_init+22>:   sd   ra,8(sp)
   0x80200024 <kern_init+24>:   jal ra,0x802004ce <memset>
   0x80200028 <kern_init+28>:   auipc   a1,0x0
   0x8020002c <kern_init+32>:   addi    a1,a1,1208
   0x80200030 <kern_init+36>:   auipc   a0,0x0
```

```
    0x80200034 <kern_init+40>:   addi    a0,a0,1232
    0x80200038 <kern_init+44>:   jal ra,0x80200058 <cprintf>
    0x8020003c <kern_init+48>:   j   0x8020003c <kern_init+48>
    0x8020003e <cputch>: addi    sp,sp,-16
    0x80200040 <cputch+2>:   sd  s0,0(sp)
    0x80200042 <cputch+4>:   sd  ra,8(sp)
    0x80200044 <cputch+6>:   mv  s0,a1
    0x80200046 <cputch+8>:   jal ra,0x8020008c <cons_putc>
```

在上面的汇编代码中，先是进入kern_entry(kern/init/entry.s),然后跳转至<kern_int>函数内部，在其中调用函数。

可以发现 0x80200038 <kern_init+44>:   jal   ra,0x80200058 调用了cprintf函数，在输出行打印了字符串(THU.CST) os is loading ...



```
PMP0: 0x0000000080000000-0x000000008001ffff (A)
PMP1: 0x0000000000000000-0xffffffffffffffff (A,R,W,X)
(THU.CST) os is loading ...
```

即计算机在操作系统内核被成功加载后，完成了对一个字符串的打印过程