

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。**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

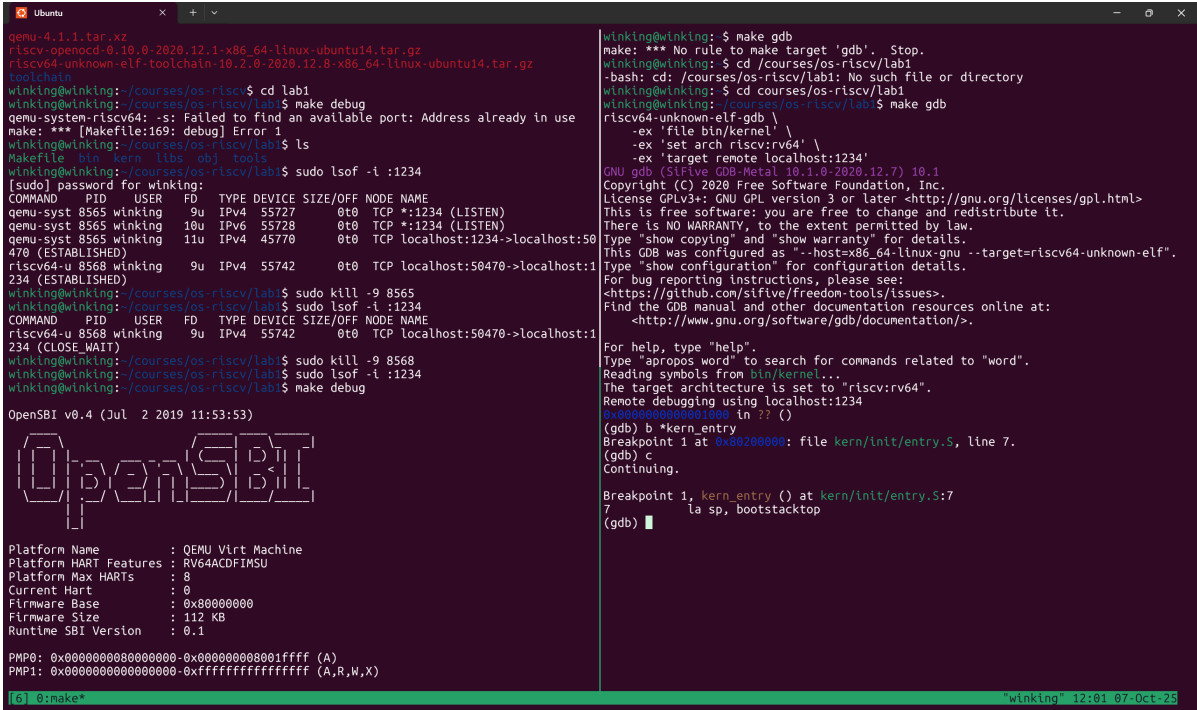
CPU加电复位后从固定复位地址开始执行也就是0x1000

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

```
qemu-4.1.1.tar.xz
riscv-openocd-0.10.0-2020.12.1-x86_64-linux-ubuntu14.tar.gz
riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-linux-ubuntu14.tar.gz
toolchain
winking@winking:~/courses/os-riscv$ cd lab1
winking@winking:~/courses/os-riscv/lab1$ make debug
qemu-system-riscv64: -s: Failed to find an available port: Address already in use
make: *** [Makefile:169: debug] Error 1
winking@winking:~/courses/os-riscv/lab1$ ls
Makefile bin kern libs obj tools
winking@winking:~/courses/os-riscv/lab1$ sudo lsof -i :1234
[sudo] password for winking:
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
qemu-syst 8565 winking 9u IPv4 55727 0t0 TCP *:1234 (LISTEN)
qemu-syst 8565 winking 10u IPv6 55728 0t0 TCP *:1234 (LISTEN)
qemu-syst 8565 winking 11u IPv4 45770 0t0 TCP localhost:1234->localhost:50
470 (ESTABLISHED)
riscv64-u 8568 winking 9u IPv4 55742 0t0 TCP localhost:50470->localhost:1
234 (ESTABLISHED)
winking@winking:~/courses/os-riscv/lab1$ sudo kill -9 8565
winking@winking:~/courses/os-riscv/lab1$ sudo lsof -i :1234
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
riscv64-u 8568 winking 9u IPv4 55742 0t0 TCP localhost:50470->localhost:1
234 (CLOSE_WAIT)
winking@winking:~/courses/os-riscv/lab1$ sudo kill -9 8568
winking@winking:~/courses/os-riscv/lab1$ sudo lsof -i :1234
winking@winking:~/courses/os-riscv/lab1$ make debug

OpenSBI v0.4 (Jul 2 2019 11:53:53)

OpenSBI

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: 0x0000000000000000-0x0000000000001fffff (A)
PMP1: 0x0000000000000000-0xffffffffffffffff (A,R,W,X)

[6] 0:make*
```

观察堆栈初始化并跟踪 OpenSBI 到内核加载过程：

```
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: 0x0000000000000000-0x0000000000001fffff (A)
PMP1: 0x0000000000000000-0xffffffffffffffff (A,R,W,X)
(THU.CST) os is loading ...

QEMU: Terminated
winking@winking:~/courses/os-riscv/lab1$ sudo lsof -i :1234
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
riscv64-u 8682 winking 9u IPv4 65201 0t0 TCP localhost:58378->localhost:1
234 (CLOSE_WAIT)
winking@winking:~/courses/os-riscv/lab1$ sudo kill -9 8682
winking@winking:~/courses/os-riscv/lab1$ sudo lsof -i :1234
winking@winking:~/courses/os-riscv/lab1$ make debug

OpenSBI v0.4 (Jul 2 2019 11:53:53)

OpenSBI

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: 0x0000000000000000-0x0000000000001fffff (A)
PMP1: 0x0000000000000000-0xffffffffffffffff (A,R,W,X)

[6] 0:make*
```

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

```
Platform Name      : QEMU Virt Machine
Platform HART Features : RV64ACDFIMSU
Platform Max HARTs   : 8
Current Hart        : 0
Firmware Base       : 0x00000000
Firmware Size       : 112 KB
Runtime SBI Version  : 0.1

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

QEMU: Terminated
winking@winking:~/courses/os-riscv/lab1$ sudo lsuf -i :1234
[sudo] password for winking:
COMMAND  PID  USER  FD   TYPE DEVICE SIZE/OFF NODE NAME
riscv64-u 8682 winking 9u  IPv4  65201      0t0  TCP localhost:58378->localhost:1234 (CLOSE_WAIT)
winking@winking:~/courses/os-riscv/lab1$ sudo kill -9 8682
winking@winking:~/courses/os-riscv/lab1$ sudo lsuf -i :1234
winking@winking:~/courses/os-riscv/lab1$ make debug

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       : 0x00000000
Firmware Size       : 112 KB
Runtime SBI Version  : 0.1

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

0x0000000000000004 in kern_entry () at kern/lnit/entry.S:7
7      la sp, bootstacktop
(gdb) info registers
ra      0x000000002  0x000000002
sp      0x00203000  0x00203000 <SBI_CONSOLE_PUTCHAR>
gp      0x0        0x0
tp      0x0001be00  0x0001be00
t0      0x00200000  2149580800
t1      0x1        1
t2      0x1        1
fp      0x0001bd90  0x0001bd90
s1      0x0001be00  2147597824
a0      0x0        0
a1      0x82200000  2183135232
a2      0x00200000  2149580800
a3      0x1        1
a4      0x800      2048
a5      0x1        1
a6      0x82200000  2183135232
a7      0x00200000  2149580800
s2      0x000095c0  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      0x00200004  0x00200004 <kern_entry+4>
dscratch Could not fetch register "dscratch"; remote failure reply 'E14'
mucounteren Could not fetch register "mucounteren"; remote failure reply 'E14'
(gdb) x/10t $pc
=> 0x00200004 <kern_entry+4>:  mv      sp,sp
0x00200008 <kern_entry+8>:  j        0x0020000a <kern_init>
0x0020000c <kern_init>:    auipc    a0,0x3
0x00200010 <kern_init+4>:  addi    a0,a0,-2
0x00200014 <kern_init+8>:  auipc    a2,0x3
0x00200018 <kern_init+12>: addi    a2,a2,-10
0x0020001c <kern_init+16>: addi    sp,sp,-16
0x00200020 <kern_init+18>: li      a1,0
0x00200024 <kern_init+20>: sub     a2,a2,a0
0x00200028 <kern_init+22>: sd      ra,0(sp)
(gdb) █
```

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。

总结：本实验中重要的知识点以及对对应得os原理中的知识点（待队友补充）

1. RISC-V加电复位和启动流程：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。

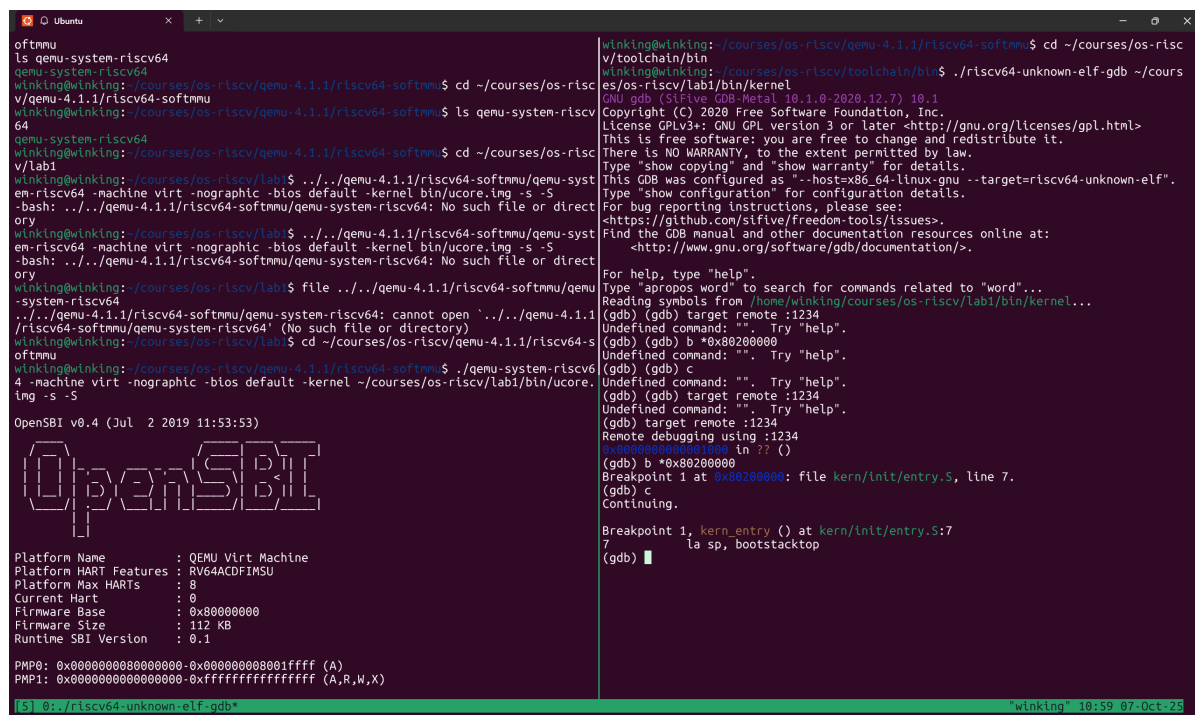
本实验未涉及(课堂见过)：

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：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



```
oftnmu
ls qemu-system-riscv64
qemu-system-riscv64
winking@winking:~/courses/os-riscv/qemu-4.1.1/riscv64-softmmu$ cd ~/courses/os-riscv/qemu-4.1.1/riscv64-softmmu
winking@winking:~/courses/os-riscv/qemu-4.1.1/riscv64-softmmu$ ls qemu-system-riscv64
qemu-system-riscv64
winking@winking:~/courses/os-riscv/qemu-4.1.1/riscv64-softmmu$ cd ~/courses/os-riscv/lab1
winking@winking:~/courses/os-riscv/lab1$ ../qemu-4.1.1/riscv64-softmmu/qemu-system-riscv64 -machine virt -nographic -bios default -kernel bin/ucore.ing -s -S
-bash: ../qemu-4.1.1/riscv64-softmmu/qemu-system-riscv64: No such file or directory
winking@winking:~/courses/os-riscv/lab1$ ../qemu-4.1.1/riscv64-softmmu/qemu-system-riscv64 -machine virt -nographic -bios default -kernel bin/ucore.ing -s -S
-bash: ../qemu-4.1.1/riscv64-softmmu/qemu-system-riscv64: No such file or directory
winking@winking:~/courses/os-riscv/lab1$ file ../qemu-4.1.1/riscv64-softmmu/qemu-system-riscv64
../qemu-4.1.1/riscv64-softmmu/qemu-system-riscv64: cannot open `../qemu-4.1.1/riscv64-softmmu/qemu-system-riscv64' (No such file or directory)
winking@winking:~/courses/os-riscv/lab1$ cd ~/courses/os-riscv/qemu-4.1.1/riscv64-softmmu
winking@winking:~/courses/os-riscv/qemu-4.1.1/riscv64-softmmu$ ./qemu-system-riscv64 -machine virt -nographic -bios default -kernel ~/courses/os-riscv/lab1/bin/ucore.ing -s -S

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

PMPO: 0x0000000000000000-0x0000000000001fffff (A)
PMP1: 0x0000000000000000-0xffffffffffffffff (A,R,W,X)

winking@winking:~/courses/os-riscv/qemu-4.1.1/riscv64-softmmu$ cd ~/courses/os-riscv/lab1/bin/kernel
GNU gdb (Sifive GDB-Metal 10.1.0-2020.12.7) 10.1
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-linux-gnu --target=riscv64-unknown-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://github.com/sifive/freedom-tools/issues>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from /home/winking/courses/os-riscv/lab1/bin/kernel...
(gdb) (gdb) target remote :1234
Undefined command: ". Try "help".
(gdb) (gdb) b *0x80200000
Undefined command: ". Try "help".
(gdb) (gdb) c
Undefined command: ". Try "help".
(gdb) (gdb) target remote :1234
Undefined command: ". Try "help".
(gdb) target remote :1234
Remote debugging using :1234
0x0000000000001000 in ?? ()
(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)
```

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。

```

0x0000000080200004 in kern_entry () at kern/init/entry.S:7
7       la sp, bootstacktop
(gdb) info register
ra          0x80000a02      0x80000a02
sp          0x80203000      0x80203000 <SBI_CONSOLE_PUTCHAR>
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          0x80200004      0x80200004 <kern_entry+4>
dscratch    Could not fetch register "dscratch"; remote failure reply 'E14'
mucounteren Could not fetch register "mucounteren"; remote failure reply 'E14'
(gdb) p &bootstacktop
$1 = (<data variable, no debug info> *) 0x80203000 <SBI_CONSOLE_PUTCHAR>

```

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)