

# 实验四-RV64内核引导

姓名:王晶晶 学号:3200104880 学院:计算机科学与技术学院 课程名称:计算机系统II

实验时间: 2021.12.6 实验地点: 紫金港机房中心 指导老师: 申文博

## 一、实验目的和要求

- 学习 RISC-V 汇编, 编写 head.S 实现跳转到内核运行的第一个 C 函数
- 学习 OpenSBI, 理解 OpenSBI 在实验中所起到的作用, 并调用 OpenSBI 提供的接口完成字符的输出。
- 学习 Makefile 相关知识, 补充项目中的 Makefile 文件, 来完成对整个工程的管理。

## 二、实验内容和原理

### 2.1 SBI 与 OpenSBI

SBI (Supervisor Binary Interface) 是 S-mode 的 Kernel 和 M-mode 执行环境之间的接口规范, 而 OpenSBI 是一个 RISC-V SBI 规范的开源实现。RISC-V 平台和 SoC 供应商可以自主扩展 OpenSBI 实现, 以适应特定的硬件配置。

在实验中, QEMU 已经内置了 OpenSBI 作为 Bootloader, 可以使用 `-bios default` 启用。如果启用, QEMU 会将 OpenSBI 代码加载到 `0x80000000` 起始处。OpenSBI 初始化完成后, 会跳转到 `0x80200000` 处 (也就是 Kernel 的起始地址)。因此, 所编译的代码需要放到 `0x80200000` 处。

### 2.2 Makefile

Makefile 可以认为是一个工程文件的编译规则, 描述了整个工程的编译和链接流程。一个工程中的源文件不计其数, 并且按类型、功能、模块分别放在若干个目录中, Makefile 定义了一系列的规则来指定, 哪些文件需要先编译, 哪些文件需要后编译, 哪些文件需要重新编译, 甚至于进行更复杂的功能操作。

Makefile带来的好处就是——“自动化编译”, 一旦写好, 只需要一个make命令, 整个

工程完全自动编译，极大的提高了软件开发的效率。make是一个命令工具，是一个解释Makefile中指令的命令工具。

## 2.3 内联汇编

内联汇编（通常由 `asm` 或者 **`asm`** 关键字引入）提供了将汇编语言源代码嵌入 C 程序的能力。

内联汇编程序模板由汇编指令、输入、输出组成。输入操作数是充当指令输入操作数使用的 C 表达式。输出操作数是将对其执行汇编指令输出的 C 表达式。

内联汇编的重要性体现在它能够灵活操作，而且可以使其输出通过 C 变量显示出来。因为它具有这种能力，所以 "asm" 可以用作汇编指令和包含它的 C 程序之间的接口。

```
__asm__ volatile (  
    "instruction1\n"  
    "instruction2\n"  
    .....  
    .....  
    "instruction3\n"  
    : [out1] "=r" (v1), [out2] "=r" (v2)  
    : [in1] "r" (v1), [in2] "r" (v2)  
    : "memory"  
);
```

## 三、主要仪器设备

- Docker in Lab3

## 四、操作方法与实现步骤

### 4.1 实验步骤

- 1、准备工程
- 2、编写head.S
- 3、完善 Makefile 脚本
- 4、补充 sbi.c

5、 puts() 和 puti()

6、 修改 defs

## 4.2 实验结果

### 4.2.1 head.S

```
.extern start_kernel

        .section .text.entry
        .globl _start
_start:
    # -----
    # - your code here -
    # -----
    #auipc sp,%hi(boot_stack_top)
    #addi sp,sp,%lo(boot_stack_top)
    #addi sp,sp,0x4

    la sp,boot_stack_top

    #auipc t0,%hi(start_kernel)
    #jalr ra,t0,%lo(start_kernel)

    call start_kernel

        .section .bss.stack
        .globl boot_stack
boot_stack:
    .space 0x1000 # <-- change to your stack size

        .globl boot_stack_top
boot_stack_top:
```

### 4.2.2 sbi.c

```

struct sbiret sbi_ecall(int ext, int fid, uint64 arg0,
    uint64 arg1, uint64 arg2,
    uint64 arg3, uint64 arg4,
    uint64 arg5)
{
    // unimplemented
    int ret1,ret2;
    __asm__ volatile(
        "mv a7,%[ext]\n"
        "mv a6,%[fid]\n"
        "mv a0,%[arg0]\n"
        "mv a1,%[arg1]\n"
        "mv a2,%[arg2]\n"
        "mv a3,%[arg3]\n"
        "mv a4,%[arg4]\n"
        "mv a5,%[arg5]\n"
        "ecall\n"
        "mv %[ret1],a0\n"
        "mv %[ret2],a1"
        :[ret1]"=r"(ret1),[ret2]"=r"(ret2)
        :[ext]"r"(ext),[fid]"r"(fid),[arg0]"r"(arg0),[arg1]"r"(arg1),[arg2]"r"(ar
            );
}

```

### 4.2.3 print.c

```

#include "print.h"
#include "sbi.h"

void puts(char *s) {
    int i=0;
    for(i=0;s[i];i++)
        sbi_ecall(0x1,0x0,s[i],0,0,0,0,0);
    return;
}

void puti(int x) {
    int a=x,i,res=1;
    while(a/=10){
        res=res*10;
    }
    a=x;
    while(res){
        sbi_ecall(0x1,0x0,a/res+'0',0,0,0,0,0);
        a=a%res;
        res=res/10;
    }
    return;
}

```

## 4.2.4 Makefile

```

C_SRC      = $(sort $(wildcard *.c))
OBJ        = $(patsubst %.c,%.o,$(C_SRC))

file = print.o
all:$(OBJ)

%.o:%.c
    ${GCC} ${CFLAG} -c $<

clean:
    $(shell rm *.o 2>/dev/null)

```

## 4.2.5 make run

Output "2021 Hello RISC-V"

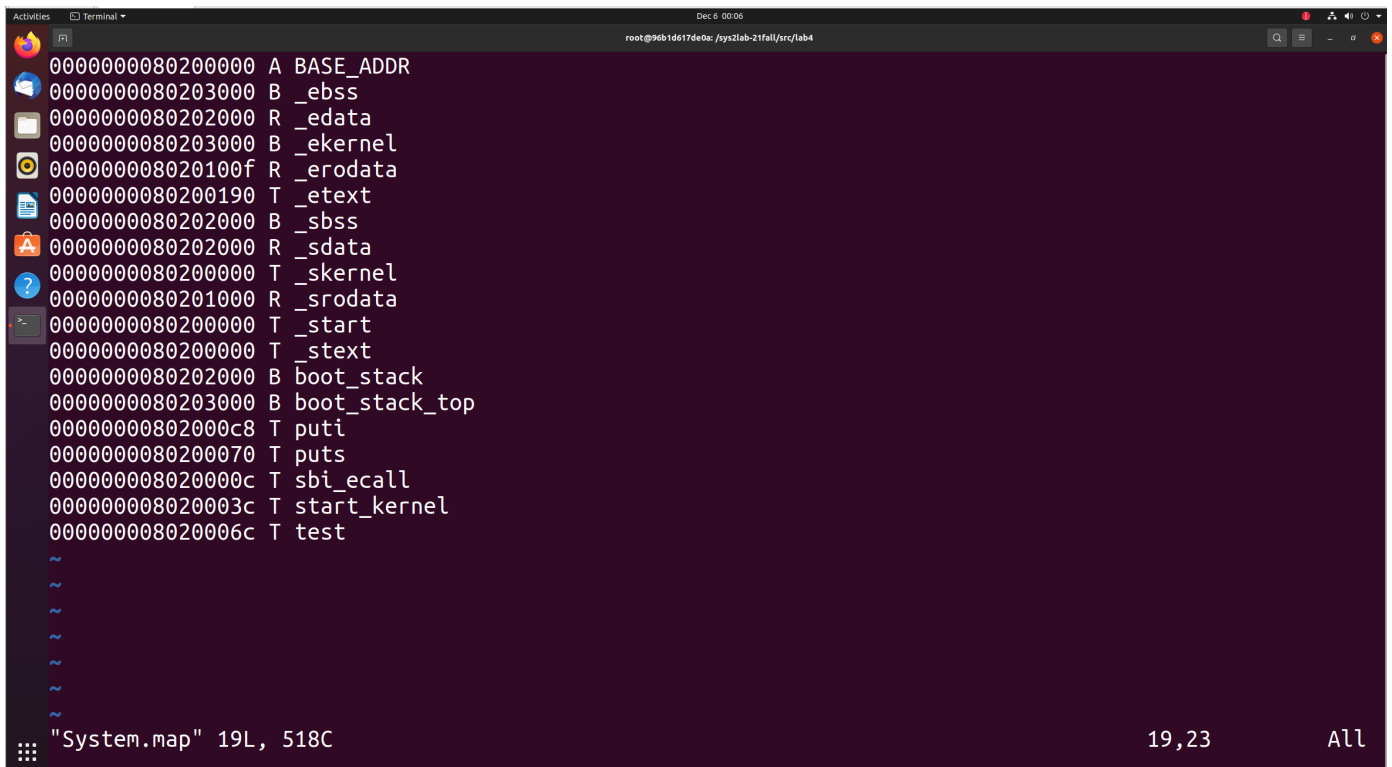
```
Activities terminal Dec 5 23:34
root@96b1d617de0a: /sys2lab-21fall/src/lab4/init
Runtime SBI Version : 0.2
Domain0 Name : root
Domain0 Boot HART : 0
Domain0 HARTs : 0*
Domain0 Region00 : 0x00000000080000000-0x0000000008001ffff ( )
Domain0 Region01 : 0x00000000000000000-0xfffffffffffff (R,W,X)
Domain0 Next Address : 0x00000000080200000
Domain0 Next Arg1 : 0x00000000087000000
Domain0 Next Mode : S-mode
Domain0 SysReset : yes

Boot HART ID : 0
Boot HART Domain : root
Boot HART ISA : rv64imafdcsv
Boot HART Features : scounteren,mcounteren,time
Boot HART PMP Count : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 54
Boot HART MHPM Count : 0
Boot HART MHPM Count : 0
Boot HART MIDELEG : 0x00000000000000222
Boot HART MEDELEG : 0x0000000000000b109
2021 Hello RISC-V
QEMU: Terminated
```

# 五、思考与心得

- 1.请总结一下 RISC-V 的 calling convention，并解释 Caller / Callee Saved Register 有什么区别？  
calling convention
  - 寄存器传递
  - 栈传递
  - 参数传递优先使用寄存器传递。  
其中默认有8个整数寄存器a0-a7和8个浮点寄存器fa0-fa7可以用，前两个（a0，a1）、（fa0，fa1）用来传递返回值，a2-a7用于传递函数参数。  
7个整数寄存器t0-t6和12个浮点寄存器ft0-ft11是临时寄存器，在调用过程中可以被修改。  
12个整数寄存器s0-s11和12个浮点寄存器fs0-fs11在调用过程后被保持不变  
Caller/Callee Saved Register
- Caller Save: 调用函数可能要将返回值放在某个确定寄存器中，在caller函数中，需要提前保存该可能改变的寄存器

- Callee Save: 被调用函数如果要修改某些特定寄存器，需要提前保存，结束后再恢复原值
- 2.编译之后，通过 System.map 查看 vmlinux.lds 中自定义符号的值。

A terminal window with a dark purple background and light blue text. The title bar at the top says "Activities Terminal" and "Dec 6 00:06". The terminal shows the output of the command "cat System.map". The output lists various symbols and their addresses, such as "0000000080200000 A BASE\_ADDR", "0000000080203000 B \_ebss", "0000000080202000 R \_edata", "0000000080203000 B \_kernel", "000000008020100f R \_erodata", "0000000080200190 T \_etext", "0000000080202000 B \_sbss", "0000000080202000 R \_sdata", "0000000080200000 T \_skernel", "0000000080201000 R \_srodata", "0000000080200000 T \_start", "0000000080200000 T \_stext", "0000000080202000 B boot\_stack", "0000000080203000 B boot\_stack\_top", "00000000802000c8 T puti", "0000000080200070 T puts", "000000008020000c T sbi\_ecall", "000000008020003c T start\_kernel", and "000000008020006c T test". There are several tilde (~) characters below the last line. At the bottom of the terminal, it says "System.map" 19L, 518C on the left, "19,23" in the middle, and "All" on the right.

```
0000000080200000 A BASE_ADDR
0000000080203000 B _ebss
0000000080202000 R _edata
0000000080203000 B _kernel
000000008020100f R _erodata
0000000080200190 T _etext
0000000080202000 B _sbss
0000000080202000 R _sdata
0000000080200000 T _skernel
0000000080201000 R _srodata
0000000080200000 T _start
0000000080200000 T _stext
0000000080202000 B boot_stack
0000000080203000 B boot_stack_top
00000000802000c8 T puti
0000000080200070 T puts
000000008020000c T sbi_ecall
000000008020003c T start_kernel
000000008020006c T test
~
~
~
~
~
~
~
"System.map" 19L, 518C                                19,23                                All
```