1) **[20pts]** `make qemu` 指令将指向makefile对应的label，该指令对应于：

```
qemu-system-riscv64 \
    -machine virt \
    -nographic \
    -bios default \
    -device loader,file=bin/ucore.bin,addr=0x80200000
```

请解释以上指令中每个参数的作用

答：qemu-system-riscv64 用于模拟一个 64 位 RISC-V 架构的机器。

-machine virt 用于选择 qemu 所支持机器中类型为 virt 的，即使用虚拟开发板。-nographic 用于禁用图形输出，并将控制台的输入输出重定向到终端。

-bios default 用于自动加载默认的 OpenSBI 固件。

-device loader , file=bin/ucore.bin,addr=0x80200000，其中-device loader 用于激活一个加载器，file=bin/ucore.bin 用于指定要加载的文件路径和名称（这里是 ucore 的二进制文件），addr=0x80200000 用于指定加载上述文件的内存地址。

2) **[20pts]** 请查阅资料，理解并解释/lab/tools/kernel.ld文件以下片段中每一行的作用（参考：https://sourceware.org/binutils/docs/ld/Scripts.html)

```
SECTIONS
{
    /* Load the kernel at this address: "." means the current address */
    . = BASE_ADDRESS;

    .text : {
        *(.text.kern_entry)
        *(.text .stub .text.* .gnu.linkonce.t.*)
    }

    PROVIDE(etext = .); /* Define the 'etext' symbol to this value */

    .rodata : {
        *(.rodata .rodata.* .gnu.linkonce.r.*)
    }

    /* Adjust the address for the data segment to the next page */
    . = ALIGN(0x1000);
```

答：SECTIONS 语句块用于告诉 linker 怎么将 input sections 和 output section 进行匹配以及描述输出文件的内存布局。

.=BASE_ADDRESS 用于设置 location counter 的值为 BASE_ADDRESS（前面定义为 0xFFFFFFFFC0200000）

.text:用于描述一个 output section,该 section 的名称是“.text”，它包含了可执行代码。其中 * 是通配符，他可以与任何文件名相匹配，而.text.kern_entry, .text, .stub, .text.*，.gnu.linkonce.t.*是一系列 input section。

PROVIDE 关键字用于定义符号 etext 的值为当前 location counter 即.text output section 的末尾，它的作用是使得当程序引用 etext 却没用进行定义时，会调用这个设定值。

.rodata 同样用于描述一个 output section，它的名称为.rodata，它包含只读数据。.rodata, .rodata.* , .gnu.linonce.r.*是一系列 input section。

ALIGN 是一个内置函数，用于返回位置指针之后的第一个满足边界对齐字节数_align 的地址值，这里实现的是设置当前 location counter 为下一页的地址以对齐下文的.data output section。

3) [10pts] 请解释 /lab/kern/init/init.c 中 main函数中`memset(edata, 0, end - edata);`的参数及语句作用。（需要读到的代码有init.c, kernel.ld)

答：memset 函数的作用是将第二个 char 类型参数的值赋给从第一个参数指针指向的地址到第三个参数所代表的长度的内存空间。函数中，若已定义__HAVE_ARCH_MEMSET 则调用__memset 函数，否则将以第一个参数指针地址开始，第三个参数为长度的内存空间，初始化为第二个参数。memset(edata, 0 ,end - edata)带入参数后的含义是将从 char 类型指针 edata 到 end 所指的内存区域初始化为‘0’。

4) [20pts] 请描述cputs()指令是如何通过sbi打印字符的。

答：sbi 中的 ecall 系统调用提供了打印单个字符的功能，其调用编号为 1。在 sbi.c 文件中，我们通过内联汇编实现了对 sbi ecall 的封装，这使得我们可以通过调用接下来定义的 sbi_console_putchar()进而调用 sbi ecall 从而实现单字符打印的函数调用。接下来我们在 console.c 文件中对 sbi_console_putchar()函数进行了进一步的简单封装，并在 stdio.c 文件中定义 cputch()函数，调用 cons_putc 同时更新计数指针 cnt。最后，cputs()通过循环遍历待打印字符串，单字符调用 cputch()函数实现多字符的任意长度字符串打印效果。

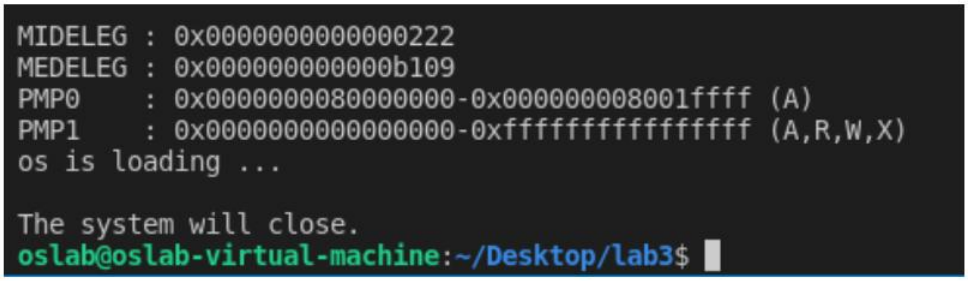
5) [30pts] 编程题 请在第三周lab.zip代码包的基础上，理解使用ecall打印字符的原理，实现一个`shutdown()`关机函数。（所有修改到的代码请截图和运行结果截图一起放在报告中）

在`while(1);`前面填加：

```
//init.c

//---------------------------------
cputs("The system will close.\n");
shutdown();
// ---------------------------------
while (1)
    ;
```

实现效果(代码不会执行到while语句)：



参考资料：riscv-sbi-doc/riscv-sbi.adoc at master · riscv-non-isa/riscv-sbi-doc (github.com)

In sbi.c

```
42  void sbi_shutdown(void) {
43      sbi_call(SBI_SHUTDOWN,0,0,0);
44  }
```

In console.h

```
7   void shutdown(void);
```

In console.c

```
26  /* shutdown - shunt down all the harts*/
27  void shutdown(void){
28      sbi_shutdown();
29  }
```

In init.c

```
30      //---------shut down-----------
31      cputs("The system will close.\n");
32      shutdown();
33      //------------------------------
```

Result

```
MIDELEG : 0x0000000000000222
MEDELEG : 0x000000000000b109
PMP0    : 0x0000000080000000-0x000000008001ffff (A)
PMP1    : 0x0000000000000000-0xffffffffffffffff (A,R,W,X)
os is loading ...

The system will close.

xjn12110714@xjn12110714-virtual-machine:~/Desktop/OS Lab/lab3$
```