浙江大学实验报告

课程名称：操作系统分析及实验 实验类型：综合型/设计性

实验项目名称：Lab 1: RISC-V 移植

学生姓名：官泽隆 专业：计算机科学与技术 学号：3180103008

电子邮件地址：3180103008@zju.edu.cn 手机：18888910213

实验日期：2020 年 10 月 29 日

一、实验目的

学习RISC-V相关知识，Makefile相关知识，M Mode到S Mode的转换等。

二、实验内容

编写head.S实现bootloader的功能，并利用Makefile来完成对整个工程的管理。

三、主要仪器设备（必填）

宿主机操作系统：Windows 10 专业版 1903
Docker Desktop for Windows，基于WSL2
Image tarbull来源
https://magiclink.teambition.com/shares/5f6ee504674e61d264aa0ef8

四、操作方法和实验步骤

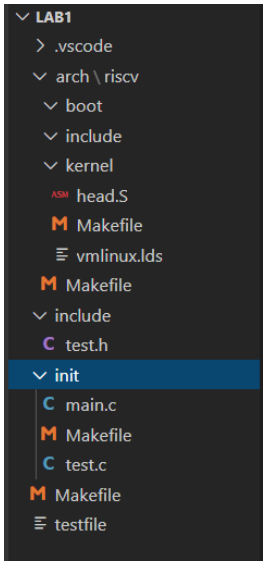
五、实验结果和分析

七、讨论、心得

实验中， 首先进行4.2，在win宿主机上组织好文件结构，win VSCode完成各级目录Makefile草稿； 然后4.1 volume挂载； 之后啃riscv以及汇编器语法参考，完成4.3 head.S，这里画了一点时间； 最后试4.4 make all，显然没能成功，进行debug，花精力作了较多修改.

4.2.1

目录结构



4.2.2

- Makefile文件中用 = 赋值时应 Watch for delayed assignment; := 赋值更符合之前编程的印象
- Makefile文件中的export 关键字向下层调用传递所有变量
- 发现顶层Makefile中，在直接用名字调用可执行程序，有 assumption

```
export RISCV=/opt/riscv
export PATH=$PATH:$RISCV/bin
```

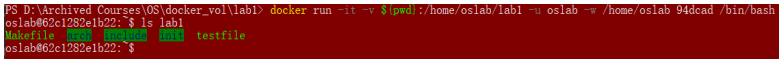
整体思路： 伪目标层次调用； 同时利用了自动推导，并重定义隐含变量： 在本实验的语境下，`<n>.o` 的目标的依赖目标的自动推导规则的先后顺序如下

> <n>.o 的目标的依赖目标会自动推导为 <n>.c ， 并且其生成命令是 $(CC) –c $(CPPFLAGS) $(CFLAGS)
>
> <n>.o 的目标的依赖目标会自动推导为 <n>.s ， 默认使用编译器 as ， 并且其生成命令是： $(AS) $(ASFLAGS) 。 <n>.s 的目标的依赖目标会自动推导为 <n>.S ， 默认使用C预编译器 cpp ， 并且其生成命令是： $(AS) $(ASFLAGS)

同时，链接Object文件时， `<n>` 目标依赖于 `<n>.o` ，通过运行C的编译器来运行链接程序生成（一般是 ld ） ，其生成命令是： `$(CC) $(LDFLAGS) <n>.o $(LOADLIBES) $(LDLIBS)`

4.1

```
cd 'D:/Archived Courses/OS/docker_vol/lab1'
docker run -it -v ${pwd}:/home/oslab/lab1 -u oslab -w /home/oslab 94dcad /bin/bash
```



说明volume挂载是成功的

4.3

其实最后写出的指令数并不多。主要是要理解 *Zicsr* 扩展提供的CSR读写接口，以及特权架构下的新增处理器状态，尤其是 当前特权模式 (which is implied) 如何用指令进入。

GNU AS 汇编写法，实验指导介绍不多。虽然我有x86汇编经验，但确实不知道这个RISC-V的汇编器接受的操作数顺序，寄存器如何指代，符号写法，常数写法，可用的伪指令，汇编器指示等等，因此当时很懵，下不了笔。找资料，最后参考了

- Chinese Reader Ref. Card; App. A
- https://github.com/riscv/riscv-asm-manual/blob/master/riscv-asm.md
- https://github.com/riscv/riscv-elf-psabi-doc/blob/master/riscv-elf.md

（建议完善实验指导）

- 指导有让设置异常处理地址。然而，我并不知道有什么我可以做的，因此就装载了代码的起始地址。实测没什么问题。
- 设置C语言调用栈环境，设置sp到哪呢？观察*linker script*其实是给出了stack_top的符号；并且根据我以前x86汇编的经验，这个符号可以认为是地址，因此猜测使用 *la* 伪指令，并试着引用了这个文件外的符号。实测可以链接，正常运行。

4.4

make实测时，发现不少路径书写错，尤其是相对路径。对Makefile调用层次也有了更深认识。最后遇到个十分weird的错误：

```
riscv64-unknown-elf-ld: ./kernel/head.o: can't link double-float modules with soft-float modules
```

怎么会用到float! 那也没有办法，调试最初像无头苍蝇一样，也不知道有没有效，调一个地方试make一次，没有解决，对问题的认识也没有加深；最后只能静下心来，一步步来。期间学习了利用objdump查看目标文件的符号和代码段反汇编。

```
riscv64-unknown-elf-objdump -x arch/riscv/kernel/head.o
riscv64-unknown-elf-objdump -d arch/riscv/kernel/head.o
riscv64-unknown-elf-objdump -d vmlinux
```

Google 报错， https://github.com/riscv/riscv-gnu-toolchain/issues/356 给了一些提示，似乎与 `-march=$ARCH and -mabi=$ABI` 体系结构特定参数有关。

反复思考整个过程，发现自己在依赖于自动推导的同时，似乎忘记了对 $ASFLAGS 的设置，默认值是空字符串。试着做

```
ASFLAGS := -march=$(ISA) -mabi=$(ABI)
```

问题解决，啊这。确实如李老师所言，因自己不认真，对整个流程不清晰，自己把自己带进坑里，陷入这种调试麻烦中。

当然，过程中也学（或者说复习）到一些东西。比如在看*objdump*时，对*li*伪指令的汇编技法加深了认识；看到汇编 *head.o* 的若干偏移为 0，而最后的image中这些偏移却又恢复正常：

```
Disassembly of section .text:

0000000000000000 <_start>:
   0:   30047073                csrci   mstatus, 8
   4:   00000297                auipc   t0, 0x0
   8:   00028293                mv      t0, t0
   c:   30529073                csrw    mtvec, t0
  10:   00000517                auipc   a0, 0x0
  14:   00050513                mv      a0, a0
  18:   34151073                csrw    mepc, a0
  1c:   000025b7                lui     a1, 0x2
  20:   8805859b                addiw   a1, a1, -1920
  24:   00001637                lui     a2, 0x1
  28:   8006061b                addiw   a2, a2, -2048
  2c:   3005b073                csrc    mstatus, a1
  30:   30062073                csrs    mstatus, a2
  34:   30200073                mret

0000000000000038 <S_start>:
  38:   10529073                csrw    stvec, t0
  3c:   00000117                auipc   sp, 0x0
  40:   00010113                mv      sp, sp
  44:   fbdff06f                j       0 <_start>
```

```
Disassembly of section .text:

0000000080000000 <_start>:
    80000000:   30047073                csrci   mstatus, 8
    80000004:   00000297                auipc   t0, 0x0
    80000008:   ffc28293                addi    t0, t0, -4 # 80000000 <_start>
    8000000c:   30529073                csrw    mtvec, t0
    80000010:   00000517                auipc   a0, 0x0
    80000014:   02850513                addi    a0, a0, 40 # 80000038 <S_start>
    80000018:   34151073                csrw    mepc, a0
    8000001c:   000025b7                lui     a1, 0x2
    80000020:   8805859b                addiw   a1, a1, -1920
    80000024:   00001637                lui     a2, 0x1
    80000028:   8006061b                addiw   a2, a2, -2048
    8000002c:   3005b073                csrc    mstatus, a1
    80000030:   30062073                csrs    mstatus, a2
    80000034:   30200073                mret

0000000080000038 <S_start>:
    80000038:   10529073                csrw    stvec, t0
    8000003c:   00008117                auipc   sp, 0x8
    80000040:   09310113                addi    sp, sp, 147 # 800080cf <_end>
    80000044:   0040006f                j       80000048 <start_kernel>
```

背后的原理其实是很深的，涉及链接与重定位(Relocation)。 虽然之前有一门软件保护技术课上有所涉猎，可是如今看到，仍然怀有一份敬畏。

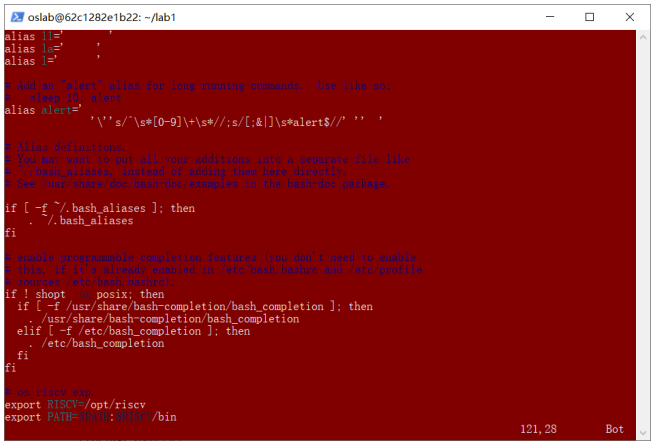
最后完成截图：

```
oslab@62c1282e1b22:~/lab1$ qemu-system-riscv64 -nographic -machine virt -kernel vmlinux
qemu-system-riscv64: warning: No -bios option specified. Not loading a firmware.
qemu-system-riscv64: warning: This default will change in a future QEMU release. Please use the -bios option to avoid breakages when this happens.
qemu-system-riscv64: warning: See QEMU's deprecation documentation for details.
Hello RISC-V!
oslab@62c1282e1b22:~/lab1$
```

后记

shell script在新开的 shell中运行，export的效力在结束后消失，所以不能用shell script自动化环境变量的设置。验收时得到老师指点，再查阅资料，学习到可以修改某几个文件来把环境变量的更改持久化：

```
vim ~/.bashrc
```


oslab@62c1282e1b22: ~/lab1
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'
Add an "alert" alias for long running commands. Use like so:
sleep 10; alert
alias alert='notify-send --urgency=low -i "$([$? = 0] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'
# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.
if [-f ~/.bash_aliases]; then
. ~/.bash_aliases
fi
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
if [-f /usr/share/bash-completion/bash_completion]; then
. /usr/share/bash-completion/bash_completion
elif [-f /etc/bash_completion]; then
. /etc/bash_completion
fi
fi
os riscv exp
export RISCV=/opt/riscv
export PATH=$PATH:$RISCV/bin
121,28 Bot