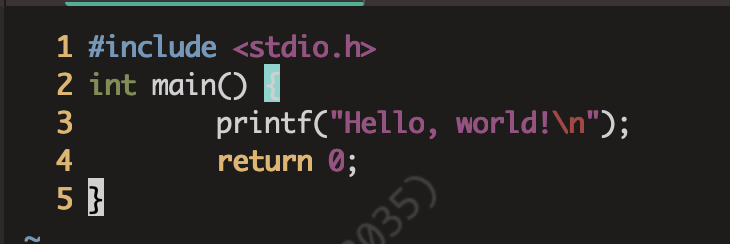
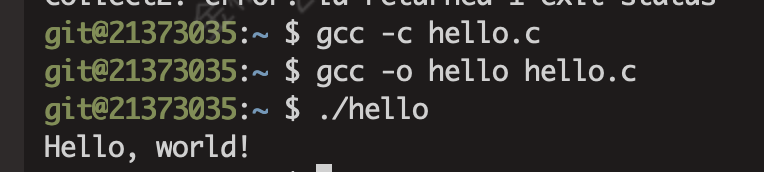
# Lab2 实验报告

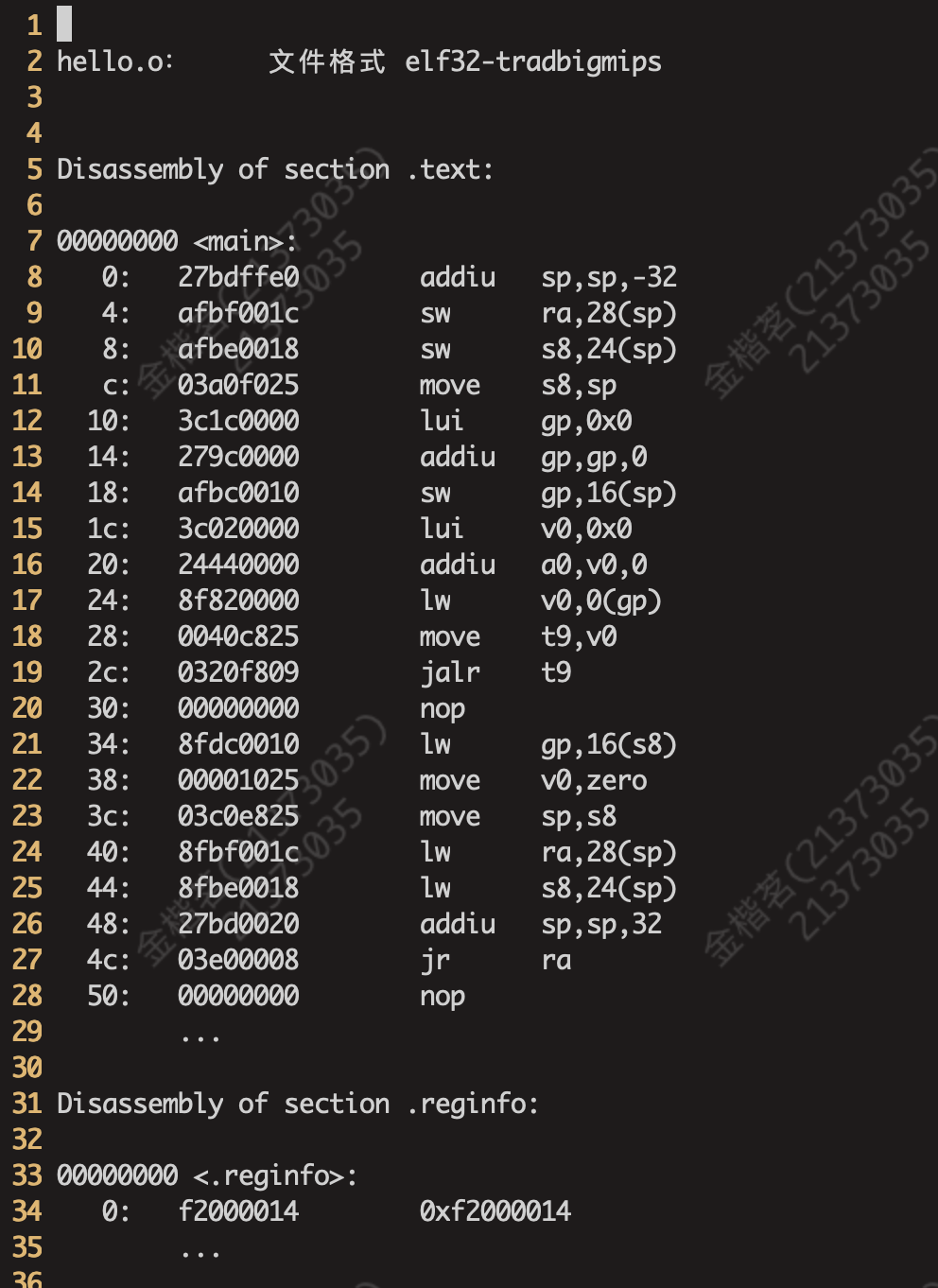
### Thinking1.1

先编辑一段简单的c语言程序，如下图所示

使用gcc编译步骤如下图所示。

使用objdump工具反汇编结果如下图所示（截取部分）。



再使用交叉编译工具链编译，命令为mips-linux-gnu-gcc -c hello.c。编译后再使用mips-linux-gnu-objdump反汇编，结果如下图所示（截取部分）。

可以看到二者在汇编语言与结构方面都有不同，前者为x86-64体系结构，后者为mips架构。

其中向Objdump传入的参数有-D和-S，-D是反汇编所有section，-S是尽可能反汇编出源代码，尤其当编译的时候指定了-g这种调试参数的时候效果比较明显。

### Thinking 0.2

(1)使用readelf解析target目录下的内核文件如下图所示。

1. 分别用自己编写的readelf与系统的readelf解析，结果如下图所示。可以看到自己的readelf无输出，系统的readelf正常解析。

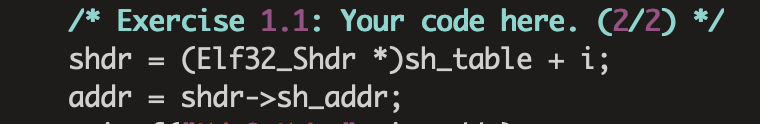
 我们在编译halleo时可以发现，编译指令是这样的：cc hello.c -o hello -m32 -static -g，说明hello是32位文件，而我们的readelf是64位的，且其只能读32位，故解析不了自身。

### Thinking 1.3

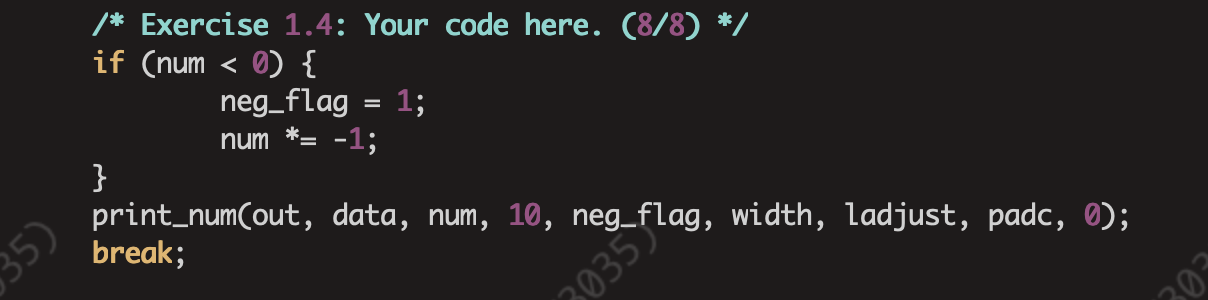
实际的操作系统启动时，首先启动bootloader程序，进入stage1并初始化硬件设备。在MIPS体系结构中，启动地址为0xBFC00000（或一个其他确定的值），将高三位抹0即得到了物理地址为0x1FC00000，然后进入stage2。

进入stage2以后，.data、.test、.bss段都被放在kseg0中，在本系统中，kernel.lds中设置好了以上三个部分，将内核调整到了正确的位置，并且通过ENTRY(\_start)设置了程序入口，之后便可正确跳转到内核启动代码。

### 难点分析

本次实验我觉得最大的难点在于涉及了很多c语言中非常底层的内容，比如指针、结构体之类的。在Exercise1.1中，我认为一个比较坑的点在要搞清楚指针的运算究竟是加了一个字节还是加了一个数据类型的大小，例如下图就是加了一个Elf\_Shdr类型的大小。与之等价的写法是(Elf\_Shdr\* sh\_table + i\*entry\_size)。

Exercise1.4我也认为颇具难度，因为其中有一些细节需要我们完整地看过一遍代码才能写对。例如下图，我们需要结合后面的print\_num函数才能明白该函数接受的num只能为正数，所以若为负数则需要取相反数。这一点虽然代码量不大，但是却需要非常细心。



### 实验体会

本次实验让我见识到了操作系统启动部分底层的实现是怎样的，加深了我对理论课上这部分内容的理解，也让我接触到了一种新的文件格式——ELF文件及其组成与工作原理，最后实现了printk函数，也让我对这个从第一天学习c语言就接触的函数——“printf”有了更深的理解。

本次实验带给我最大的感触就是指针的重要性，想当初大一学习c语言时老师就曾提醒我们指针的重要性，并且还说过“不会指针就相当于不会c语言”。起初我还有所不解，好像在编程中不用指针也完全可以进行，但到今天我才意识到学好指针的重要性，因为在操作系统的一些c语言源码中，指针是随处可见的，而且也是十分方便的。Lab1正好带我复习了c语言的指针相关内容，收获满满。

Lab1相比Lab0来说工程量大了很多，往往需要填的代码就一行，但是需要看上好几十行的代码才能填对，我在完成lab1的过程中有几次急于求成，没有完全理解代码就贸然作答，导致测试报错，这也是给我一个教训，相信今后工程量会越来越大，所以下笔之前一定要谨慎再谨慎。