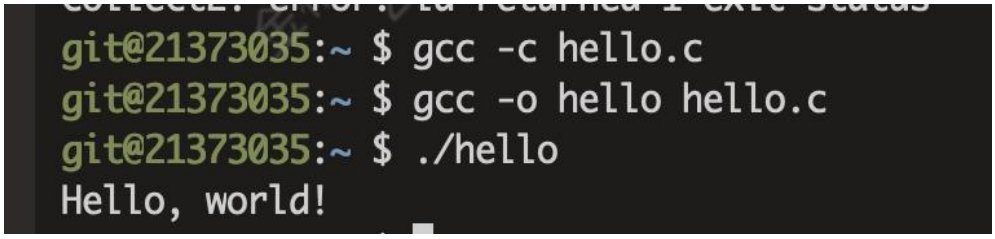
# Lab2 实验报告

## Thinking1.1

先编辑一段简单的 c 语言程序，如下图所示

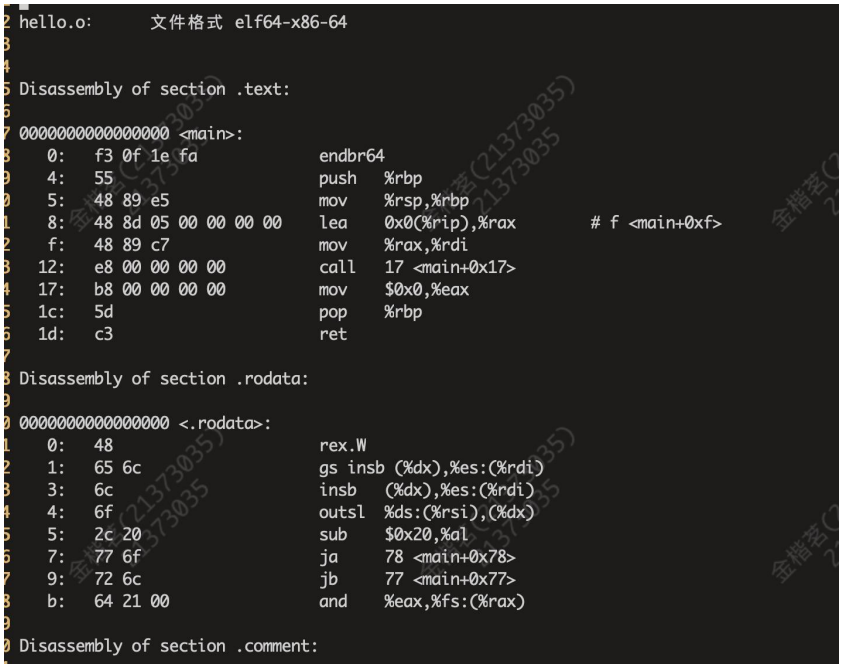
```c
#include <stdio.h>
int main() {
        printf("Hello, world!\n");
        return 0;
}
```

使用 gcc 编译步骤如下图所示。



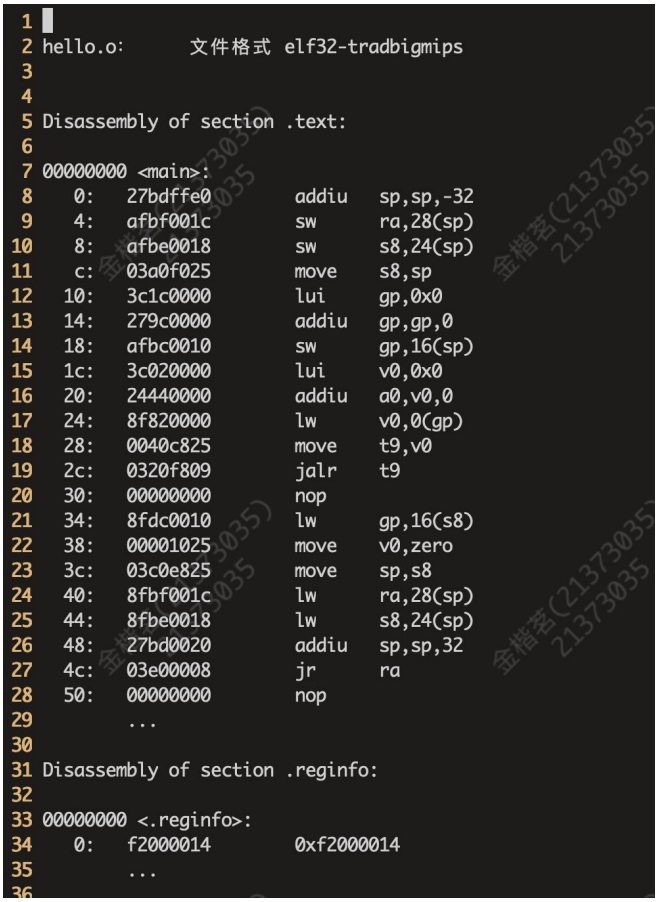
```
git@21373035:~ $ gcc -c hello.c
git@21373035:~ $ gcc -o hello hello.c
git@21373035:~ $ ./hello
Hello, world!
```

使用 objdump 工具反汇编结果如下图所示（截取部分）。



```
hello.o:     文件格式 elf64-x86-64


Disassembly of section .text:

0000000000000000 <main>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 8d 05 00 00 00 00    lea    0x0(%rip),%rax        # f <main+0xf>
   f:   48 89 c7                mov    %rax,%rdi
  12:   e8 00 00 00 00          call   17 <main+0x17>
  17:   b8 00 00 00 00          mov    $0x0,%eax
  1c:   5d                      pop    %rbp
  1d:   c3                      ret

Disassembly of section .rodata:

0000000000000000 <.rodata>:
   0:   48                      rex.W
   1:   65 6c                   gs insb (%dx),%es:(%rdi)
   3:   6c                      insb   (%dx),%es:(%rdi)
   4:   6f                      outsl  %ds:(%rsi),(%dx)
   5:   2c 20                   sub    $0x20,%al
   7:   77 6f                   ja     78 <main+0x78>
   9:   72 6c                   jb     77 <main+0x77>
   b:   64 21 00                and    %eax,%fs:(%rax)

Disassembly of section .comment:
```

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）。



```
hello.o:     文件格式 elf32-tradbigmips


Disassembly of section .text:

00000000 <main>:
   0:   27bdffe0        addiu   sp,sp,-32
   4:   afbf001c        sw      ra,28(sp)
   8:   afbe0018        sw      s8,24(sp)
   c:   03a0f025        move    s8,sp
  10:   3c1c0000        lui     gp,0x0
  14:   279c0000        addiu   gp,gp,0
  18:   afbc0010        sw      gp,16(sp)
  1c:   3c020000        lui     v0,0x0
  20:   24440000        addiu   a0,v0,0
  24:   8f820000        lw      v0,0(gp)
  28:   0040c825        move    t9,v0
  2c:   0320f809        jalr    t9
  30:   00000000        nop
  34:   8fdc0010        lw      gp,16(s8)
  38:   00001025        move    v0,zero
  3c:   03c0e825        move    sp,s8
  40:   8fbf001c        lw      ra,28(sp)
  44:   8fbe0018        lw      s8,24(sp)
  48:   27bd0020        addiu   sp,sp,32
  4c:   03e00008        jr      ra
  50:   00000000        nop
        ...

Disassembly of section .reginfo:

00000000 <.reginfo>:
   0:   f2000014        0xf2000014
        ...
```

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。

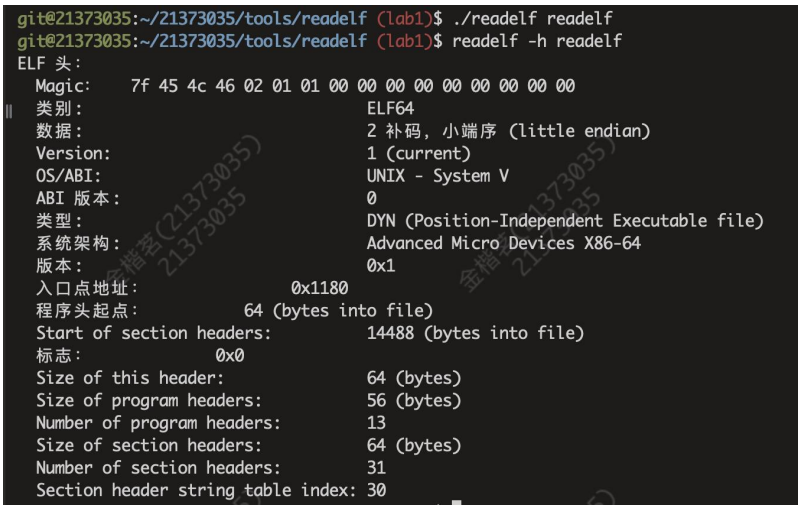
其中向 Objdump 传入的参数有-D 和-S，-D 是反汇编所有 section，-S 是尽可能反汇编出源代码，尤其当编译的时候指定了-g 这种调试参数的时候效果比较明显。

# Thinking 0.2

(1)使用 readelf 解析 target 目录下的内核文件如下图所示。

```
git@21373035:~/21373035/tools/readelf (lab1)$ ./readelf ../../target/mos
0:0x0
1:0x0
2:0x0
3:0x0
4:0x80010000
5:0x80012480
6:0x80012498
7:0x800124b0
8:0x0
9:0x0
10:0x0
11:0x0
12:0x0
13:0x0
14:0x0
15:0x0
16:0x0
```

(2) 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。

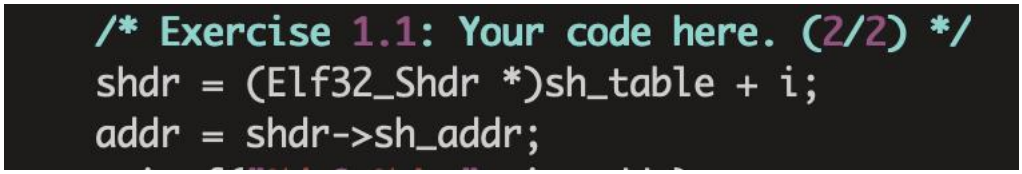
## Thinking 1.3

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。

```
/* Exercise 1.4: Your code here. (8/8) */
if (num < 0) {
        neg_flag = 1;
        num *= -1;
}
print_num(out, data, num, 10, neg_flag, width, ladjust, padc, 0);
break;
```

# 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。