Lab1

> 班级：222111
>
> 学号：22373340
>
> 姓名：詹佳博

思考题

Thinking 1.1

思考框架：

```
git@22373340:~/Lab1_Thinking $ tree
.
├── mips
│   ├── hello.c
│   ├── hello.E
│   ├── hello.exe
│   ├── hello.exeobjdump
│   ├── hello.o
│   └── hello.objdump
└── x86
    ├── hello.c
    ├── hello.E
    ├── hello.exe
    ├── hello.exeobjdump
    ├── hello.o
    └── hello.objdump
```

- `x86`：

```
git@22373340:~/Lab1_Thinking/x86 $ gcc -c hello.c
git@22373340:~/Lab1_Thinking/x86 $ objdump -DS hello.o > hello.objdump
git@22373340:~/Lab1_Thinking/x86 $ gcc -o hello.exe hello.c
git@22373340:~/Lab1_Thinking/x86 $ objdump -DS hello.exe > hello.exeobjdump
```

  - 只编译不链接的解析（反汇编）结果（`./x86/hello.objdump`）：

```
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
```

  - 编译并链接的解析（反汇编）结果（`./x86/hello.exeobjdump`）：

```
0000000000001149 <main>:
    1149:   f3 0f 1e fa             endbr64
    114d:   55                      push   %rbp
    114e:   48 89 e5                mov    %rsp,%rbp
    1151:   48 8d 05 ac 0e 00 00    lea    0xeac(%rip),%rax        # 2004 <_IO_stdin_used+0x4>
    1158:   48 89 c7                mov    %rax,%rdi
    115b:   e8 f0 fe ff ff          call   1050 <puts@plt>
    1160:   b8 00 00 00 00          mov    $0x0,%eax
    1165:   5d                      pop    %rbp
    1166:   c3                      ret
```

  - 注意到 `call` 指令在链接之后**被填入地址**了。

- `mips`：

```
git@22373340:~/Lab1_Thinking/mips $ mips-linux-gnu-gcc -c hello.c
git@22373340:~/Lab1_Thinking/mips $ mips-linux-gnu-objdump -DS hello.o > hello.objdump
git@22373340:~/Lab1_Thinking/mips $ mips-linux-gnu-gcc -o hello.exe hello.c
git@22373340:~/Lab1_Thinking/mips $ mips-linux-gnu-objdump -DS hello.exe > hello.exeobjdump
```

- 只编译不链接的解析（反汇编）结果（`./mips/hello.objdump`）：

```
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
```

- 编译并链接的解析（反汇编）结果（`./mips/hello.exeobjdump`）：

```
004006e0 <main>:
  4006e0:   27bdffe0    addiu   sp,sp,-32
  4006e4:   afbf001c    sw  ra,28(sp)
  4006e8:   afbe0018    sw  s8,24(sp)
  4006ec:   03a0f025    move    s8,sp
  4006f0:   3c1c0042    lui gp,0x42
  4006f4:   279c9010    addiu   gp,gp,-28656
  4006f8:   afbc0010    sw  gp,16(sp)
  4006fc:   3c020040    lui v0,0x40
  400700:   24440830    addiu   a0,v0,2096
  400704:   8f828030    lw  v0,-32720(gp)
  400708:   0040c825    move    t9,v0
  40070c:   0320f809    jalr    t9
  400710:   00000000    nop
  400714:   8fdc0010    lw  gp,16(s8)
  400718:   00001025    move    v0,zero
  40071c:   03c0e825    move    sp,s8
  400720:   8fbf001c    lw  ra,28(sp)
  400724:   8fbe0018    lw  s8,24(sp)
  400728:   27bd0020    addiu   sp,sp,32
  40072c:   03e00008    jr  ra
  400730:   00000000    nop
```

- 注意到在第 5 - 10 行的指令在链接之后**被填入地址**了。

- 解释其中向 `objdump` 传入的参数的含义：即 `objdump -D -S` 。
  - 其中 `-D(--disassemble-all)` 代表从 objfile 中反汇编所有指令机器码的 section；
  - 其中 `-S(--source)` 代表尽可能反汇编出源代码。

Thinking 1.2

- 执行 `./readelf`
  - 

```
git@22373340:~/22373340/tools/readelf (lab1)$ ./readelf ../../target/mos
0:0x0
1:0x80020000
2:0x800218f0
3:0x80021908
4:0x80021920
5:0x0
6:0x0
7:0x0
8:0x0
9:0x0
10:0x0
11:0x0
12:0x0
13:0x0
14:0x0
15:0x0
16:0x0
17:0x0
```

- 使用 `readelf -h`

```
git@22373340:~/22373340/tools/readelf (lab1)$ readelf -h ./readelf
ELF 头:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  类别:                              ELF64
  数据:                              2 补码，小端序 (little endian)
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI 版本:                          0
  类型:                              DYN (Position-Independent Executable file)
  系统架构:                          Advanced Micro Devices X86-64
  版本:                              0x1
  入口点地址:               0x1180
  程序头起点:          64 (bytes into file)
  Start of section headers:          14488 (bytes into file)
  标志:             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
git@22373340:~/22373340/tools/readelf (lab1)$ readelf -h ./hello
ELF 头:
  Magic:   7f 45 4c 46 01 01 01 03 00 00 00 00 00 00 00 00
  类别:                              ELF32
  数据:                              2 补码，小端序 (little endian)
  Version:                           1 (current)
  OS/ABI:                            UNIX - GNU
  ABI 版本:                          0
  类型:                              EXEC (可执行文件)
  系统架构:                          Intel 80386
  版本:                              0x1
  入口点地址:               0x8049600
  程序头起点:          52 (bytes into file)
  Start of section headers:          746252 (bytes into file)
  标志:             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         8
  Size of section headers:           40 (bytes)
  Number of section headers:         35
  Section header string table index: 34
```

- 结合 `Makefile` 可知，对 `readelf` 和 `hello` 文件的编译命令不同。可知对**后者**的参数多加了 `-m32 -static -g`。分析可知，`hello` 为 32 位代码，而 `readelf` 为 **64** 位代码。阅读 `readelf.c` 可知，我们所编写的程序只能解析 `Elf32`，即 32 位代码。

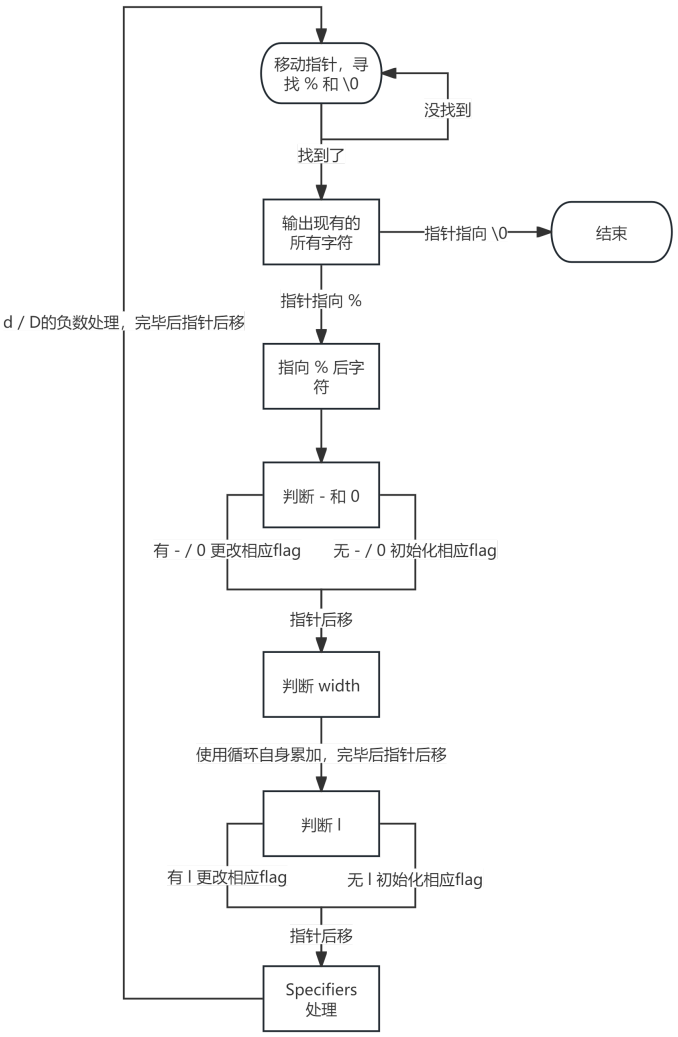
Thinking 1.3

- MOS 操作系统的目标是在 QEMU 模拟器上运行，故而启动这个过程被大大简化了。QEMU 支持加载 ELF 内核，所以启动流程被简化为**加载内核到内存**，之后**跳转到内核的入口**，启动就完成了。
- 故而，实验操作系统先将内核入口按照内存布局图放置，通过所编写的 `kernel.lds` 文件，将内核调整到正确的位子上，即**加载内核到内存**（由我们编写）；之后，在该文件头有两行代码，第一行规定架构为 mips，第二行规定程序的入口为 _start 这个函数，此时就可以**跳转到内核的入口**（由操作系统运行），也就是该函数。

难点分析

1. 在完成 `readelf.c` 函数时，对传入的 void *binary 一开始不理解，后来观察 ELF 结构，了解到 binary 指向 ELF 头，当我们想访问节头表时候，只需要将该指针往后移动到节头表位置即可。这个移动的距离只需要**调用 binary 指向的 ELF 头中所存储的位移距离**即可。移动到该位置之后，即可对指针进行强制类型转换，变为节头表，从而进一步访问其指向内存的内容，如 address 等。
2. 在完成 `Exercise 1.3` 时，要设置 sp 寄存器到内核栈空间。我编写的代码将其设置到了该空间的栈顶。一开始并不是很理解，后来想到栈的增长方向是向下增长的，指向栈顶也就可以防止栈溢出等。

3. `Exercise 1.4` 的逻辑框架复杂，用流程图表示如下：



实验体会

1. 在完成 `Thinking 1.2` 时，误判断影响原因是两个程序的类型不同。一个是EXEC，一个是DYN，后来知道了这两个的区别是是否使用了静态库的区别。后者更为灵活，可能不包含程序执行的所有代码。
2. 在完成 `Exercise 1.4` 时，第一次编写没有时刻初始化各个 flag 的值。这是个人疏忽，在编写时没有理清思路。最后还是通过流程图来理清编写过程。流程图对复杂的逻辑函数编写有着非常重要的作用。
3. 在做实验的 extra 的时候，while的逻辑循环判断错误了，导致丢分。以后得仔细看看自己逻辑判断是否正确。