

# Lab 1

## 思考题

### Thinking 1.1

- **-D: --disassemble-all** , 表示以反汇编的形式显示目标文件的内容。
- **-S: --source** , 表示在输出文件的反汇编代码之后, 同时显示源代码和汇编代码之间的对应关系。

编译与解析过程:

```
$ gcc -E Hello.c > Hello.pre
$ gcc -c Hello.c
$ objdump -DS Hello.o > Hello.asm
$ gcc -o Hello Hello.c
$ objdump -DS Hello > Hello_link.asm
$ mips-linux-gnu-gcc -E Hello_mips.c > Hello_mips.pre
$ mips-linux-gnu-gcc -c Hello_mips.c
$ mips-linux-gnu-objdump -DS Hello_mips.o > Hello_mips.asm
$ mips-linux-gnu-gcc -o Hello_mips Hello_mips.c
$ mips-linux-gnu-objdump -DS Hello_mips > Hello_link_mips.asm
$ |
```

MIPS 交叉编译工具链结果:

```
1 Hello_mips.asm 2 Hello_link_mips...
2 Hello_mips.o: 文件格式 elf32-tradbigmips
3
4
5 Disassembly of section .text:
6
7 00000000 <main>:
8 0: 27bdffe0 addiu sp,sp,-32
9 4: afbf001c sw ra,28(sp)
10 8: afbe0018 sw s8,24(sp)
11 c: 03a0f025 move s8,sp
12 10: 3c1c0000 lui gp,0x0
13 14: 279c0000 addiu gp,gp,0
14 18: afbc0010 sw gp,16(sp)
15 1c: 3c020000 lui v0,0x0
16 20: 24440000 addiu a0,v0,0
17 24: 8f820000 lw v0,0(gp)
18 28: 0040c825 move t9,v0
19 2c: 0320f809 jalr t9
20 30: 00000000 nop
21 34: 8fdc0010 lw gp,16(s8)
22 38: 00001025 move v0,zero
23 3c: 03c0e825 move sp,s8
24 40: 8fbf001c lw ra,28(sp)
25 44: 8fbe0018 lw s8,24(sp)
26 48: 27bd0020 addiu sp,sp,32
27 4c: 03e00008 jr ra
28 50: 00000000 nop
29 ...
30
31 Disassembly of section .reginfo:
32
33 00000000 <.reginfo>:
34 0: f2000014 0xf2000014
35 ...
36
37 Disassembly of section .MIPS.abiflags:
38
```

NORMAL Hello\_mips.asm utf-8 < > asm Top

```
1 Hello_mips.asm | Hello_link_mips...
2 Hello_mips: 文件格式 elf32-tradbigmips
3
4
5 Disassembly of section .interp:
6
7 00400194 <.interp>:
8 400194: 2f6c6962 sltiu t4,k1,26978
9 400198: 2f6c642e sltiu t4,k1,25646
10 40019c: 736f2e31 0x736f2e31
11 ...
12
13 Disassembly of section .MIPS.abiflags:
14
15 004001a8 <.MIPS.abiflags>:
16 4001a8: 00002002 srl a0,zero,0x0
17 4001ac: 01010005 lsa zero,t0,at,0x1
18 ...
19
20 Disassembly of section .reginfo:
21
22 004001c0 <.reginfo>:
23 4001c0: b20000f6 0xb20000f6
24 ...
25 4001d4: 00419010 0x419010
26
27 Disassembly of section .note.gnu.build-id:
28
29 004001d8 <.note.gnu.build-id>:
30 4001d8: 00000004 silv zero,zero,zero
31 4001dc: 00000014 0x14
32 4001e0: 00000003 sra zero,zero,0x0
33 4001e4: 474e5500 bz.w $w14,4155e8 <_end+0x4588>
34 4001e8: e3760032 sc s6,50(k1)
35 4001ec: 28c123a9 slti at,a2,9129
36 4001f0: 756ce088 jalx 5b38220 <_gp+0x571f210>
37 4001f4: d60dacb4 ldcl $f13,-21324(s0)
38 4001f8: a29a6f10 sb k0,28432(s4)
39
40 NORMAL Hello_link_mips.asm utf-8 < < asm Top
```

## Thinking 1.2

```
git@21371163:~/21371163/tools (lab1)$ cd ..
git@21371163:~/21371163 (lab1)$ ./tools/readelf/readelf target/mos
0:0x0
1:0x80010000
2:0x80011f20
3:0x80011f38
4:0x80011f50
5:0x80012190
6:0x0
7:0x0
8:0x0
9:0x0
10:0x0
11:0x0
12:0x0
13:0x0
14:0x0
15:0x0
16:0x0
17:0x0
git@21371163:~/21371163 (lab1)$
```

readelf 程序为 64 位，而其本身只能解析 32 位程序。

## Thinking 1.3

在MIPS体系结构中，启动入口地址为 0xBFC00000，这是硬件逻辑规定的地址，但这并不意味着内核必须放置在这个地址处才能被正确地启动。实际上，内核可以放置在内存的任何位置，只要在启动过程中正确地设置好启动地址即可。

GXemul 已经提供了 bootloader 的引导（启动）功能。MOS 操作系统不需要再实现 bootloader 的功能。在 MOS 操作系统的运行第一行代码前，我们就已经拥有一个正常的程序运行环境，内存和一些外围设备都可以正常使用。GXemul 支持加载 ELF 格式内核，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。

## 难点分析

`readelf` 与 `printk` 只要掌握基本的 c 语言知识便较为简单，主要难点在于对操作系统内存位置分配的理解。

## 实验感想

这次实验中我学到了 `elf` 格式文件的相关知识，也了解了操作系统的内存分配。实现 `printf` 功能的要求，其实我在上学期任程设助教时为了出题就尝试过一些，于是便比较轻松地完成了。（但是当时出的题只有一个同学过了）