Lab1实验报告

一、实验思考题

Thinking 1.1

-D/ --disassemble-all : 反汇编所有section

-S/--source : 尽可能反汇编出源代码

使用交叉编译器mips_4KC重复过程：

```
/OSLAB/compiler/usr/bin/mips_4KC-gcc -c fibo.c
/OSLAB/compiler/usr/bin/mips_4KC-ld -o fibo_mips fibo.o
/OSLAB/compiler/usr/bin/mips_4KC-objdump -DS fibo.o > mips_fibo_o.txt
/OSLAB/compiler/usr/bin/mips_4KC-objdump -DS fibo_mips > mips_fibo_exe.txt
```

得到mips_fibo_o.txt：

```

fibo.o:     file format elf32-tradbigmips

Disassembly of section .text:

00000000 <fib>:
   0:   3c1c0000    lui gp,0x0
   4:   279c0000    addiu   gp,gp,0
   8:   0399e021    addu    gp,gp,t9
   c:   27bdffd0    addiu   sp,sp,-48
  10:   afbf0028    sw  ra,40(sp)
  14:   afbe0024    sw  s8,36(sp)
  18:   afb00020    sw  s0,32(sp)
  1c:   03a0f021    move    s8,sp
  20:   afbc0010    sw  gp,16(sp)
  24:   afc40030    sw  a0,48(s8)
  28:   8fc20030    lw  v0,48(s8)
  2c:   10400005    beqz    v0,44 <fib+0x44>
  30:   00000000    nop
  34:   8fc30030    lw  v1,48(s8)
  38:   24020001    li  v0,1
  3c:   14620005    bne v1,v0,54 <fib+0x54>
  40:   00000000    nop
  44:   24020001    li  v0,1
  48:   afc20018    sw  v0,24(s8)
  4c:   10000012    b   98 <fib+0x98>
  50:   00000000    nop
  54:   8fc20030    lw  v0,48(s8)
  58:   2442ffff    addiu   v0,v0,-1
  5c:   00402021    move    a0,v0
  60:   8f990000    lw  t9,0(gp)
  64:   0320f809    jalr    t9
  68:   00000000    nop
  6c:   8fdc0010    lw  gp,16(s8)
  70:   00408021    move    s0,v0
```

```
  74:   8fc20030    lw  v0,48(s8)
  78:   2442fffe    addiu   v0,v0,-2
  7c:   00402021    move    a0,v0
  80:   8f990000    lw  t9,0(gp)
  ...
```

得到mips_fibo_exe.txt:

```
fibo_mips:     file format elf32-tradbigmips

Disassembly of section .reginfo:

00400094 <.reginfo>:
  400094:   f2010010    0xf2010010
    ...
  4000a8:   10007ff0    b   42006c <main+0x1ff04>
Disassembly of section .text:

004000b0 <fib>:
  4000b0:   3c1c0fc0    lui gp,0xfc0
  4000b4:   279c7f40    addiu   gp,gp,32576
  4000b8:   0399e021    addu    gp,gp,t9
  4000bc:   27bdffd0    addiu   sp,sp,-48
  4000c0:   afbf0028    sw  ra,40(sp)
  4000c4:   afbe0024    sw  s8,36(sp)
  4000c8:   afb00020    sw  s0,32(sp)
  4000cc:   03a0f021    move    s8,sp
  4000d0:   afbc0010    sw  gp,16(sp)
  4000d4:   afc40030    sw  a0,48(s8)
  4000d8:   8fc20030    lw  v0,48(s8)
  4000dc:   10400005    beqz    v0,4000f4 <fib+0x44>
  4000e0:   00000000    nop
  4000e4:   8fc30030    lw  v1,48(s8)
  4000e8:   24020001    li  v0,1
  4000ec:   14620005    bne v1,v0,400104 <fib+0x54>
  4000f0:   00000000    nop
  4000f4:   24020001    li  v0,1
  4000f8:   afc20018    sw  v0,24(s8)
  4000fc:   10000012    b   400148 <fib+0x98>
  400100:   00000000    nop
  400104:   8fc20030    lw  v0,48(s8)
  400108:   2442ffff    addiu   v0,v0,-1
  40010c:   00402021    move    a0,v0
  400110:   8f99802c    lw  t9,-32724(gp)
  400114:   0320f809    jalr    t9
  400118:   00000000    nop
```

Thinking 1.2

testELF文件采用小端存储，而vmlinux采用大端存储，因而与我们自己的readelf不匹配，会

输出`Segmentation fault (core dumped)`

Thinking 1.3

GXemul仿真器提供了bootloader全部功能，而跳转到内核入口的操作会在bootloader的stage2中完成。因而我们可以不管启动的事情，直接设置内核入口。

Thinking 1.4

加载的时候，避免页面共享和页面冲突：若当前页已被其他段占用，则在下一页加载。

（看指导书我只知道用LinkerScript可以手动分配加载地址，其他就啥也不懂了...）

Thinking 1.5

内核入口：`0x8001_0000`

main函数入口：`0x8001_0050`

```
Symbol table '.symtab' contains 37 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 80010000     0 SECTION LOCAL  DEFAULT    1
     2: 80010ae0     0 SECTION LOCAL  DEFAULT    2
     3: 80010af8     0 SECTION LOCAL  DEFAULT    3
     4: 80010ba0     0 SECTION LOCAL  DEFAULT    4
     5: 80010db0     0 SECTION LOCAL  DEFAULT    5
     6: 80010db0     0 SECTION LOCAL  DEFAULT    6
     7: 80018db0     0 SECTION LOCAL  DEFAULT    7
     8: 00000000     0 SECTION LOCAL  DEFAULT    8
     9: 00000000     0 SECTION LOCAL  DEFAULT    9
    10: 00000000     0 SECTION LOCAL  DEFAULT   10
    11: 00000000     0 SECTION LOCAL  DEFAULT   11
    12: 00000000     0 SECTION LOCAL  DEFAULT   12
    13: 00000000     0 SECTION LOCAL  DEFAULT   13
    14: 00000000     0 FILE    LOCAL  DEFAULT  ABS start.S
    15: 80010db0     0 NOTYPE  LOCAL  DEFAULT    6 KERNEL_STACK
    16: 80010038     0 NOTYPE  LOCAL  DEFAULT    1 loop
    17: 00000000     0 FILE    LOCAL  DEFAULT  ABS main.c
    18: 00000000     0 FILE    LOCAL  DEFAULT  ABS init.c
    19: 00000000     0 FILE    LOCAL  DEFAULT  ABS console.c
    20: 00000000     0 FILE    LOCAL  DEFAULT  ABS print.c
    21: 80010ba0    25 OBJECT  LOCAL  DEFAULT    4 theFatalMsg
    22: 00000000     0 FILE    LOCAL  DEFAULT  ABS printf.c
    23: 800109a0   152 FUNC    LOCAL  DEFAULT    1 myoutput
    24: 80010130  1332 FUNC    GLOBAL DEFAULT    1 lp_Print
    25: 80010a78   100 FUNC    GLOBAL DEFAULT    1 _panic
    26: 800100e8    72 FUNC    GLOBAL DEFAULT    1 printstr
    27: 80010a38    64 FUNC    GLOBAL DEFAULT    1 printf
    28: 80010800   408 FUNC    GLOBAL DEFAULT    1 PrintNum
    29: 800100c0    20 FUNC    GLOBAL DEFAULT    1 printcharc
    30: 80018db0     0 NOTYPE  GLOBAL DEFAULT  ABS end
    31: 80010000    68 FUNC    GLOBAL DEFAULT    1 _start
    32: 800106e0   288 FUNC    GLOBAL DEFAULT    1 PrintString
    33: 80010050    52 FUNC    GLOBAL DEFAULT    1 main
    34: 80010664   124 FUNC    GLOBAL DEFAULT    1 PrintChar
    35: 800100d4    20 FUNC    GLOBAL DEFAULT    1 halt
    36: 80010090    44 FUNC    GLOBAL DEFAULT    1 mips_init
```

在设置完栈指针后，使用`jal main`跳转到main函数；

跨文件函数类似，用栈存储相关需保护的数据，然后jal跳转到相应函数。

Thinking 1.6

```
mtc0    zero, CP0_STATUS
/* 向SR寄存器写零，从而禁止中断异常 */
/* SR[15:8]置零屏蔽中断信号，SR[0]置零屏蔽全局中断 */
```

```
mfc0    t0, CP0_CONFIG
and     t0, ~0x7    /* 0xffff_fffa  将Config[2] Config[0]置0 */
ori     t0, 0x2     /* 0x0000_0002  将Config[1]置1 */
mtc0    t0, CP0_CONFIG
/* 向Config[2:0]写0b010使核心态不经过Cache */
/* Config[2:0]决定kseg0区是否经过Cache以及去确切行为 */
```

二、实验难点

Exercise1.1-1.4：理清每一步操作在vmlinux配置上的意义和作用，对操作系统有切实的认识，并认清用gxemul模拟与现实的不同。

Exercise1.5：C语言使用，`va_list`和不定长参数；应该对print.c和printf.c在整个操作系统的位置和层次有认识

三、体会与感想

有举步维艰的感觉，需要查看许多资料后，形成一定认识才能动手。前置知识匮乏也导致一些理解没能太到位。总体上一直有种云里雾里的感觉。

（时间太久远，记不太清了。。）

四、指导书建议

没精力细细体会了，感觉主线脉络不是很明晰，前几个Exercise做得挺迷茫，回头看时才找到和总操作系统的联系。

小问题：

P60：第二个代码块下面第二行，应该是“根据x86指令的特点”

P80：`myoutput()`的第一个参数好像没使用，有些疑惑，想解释