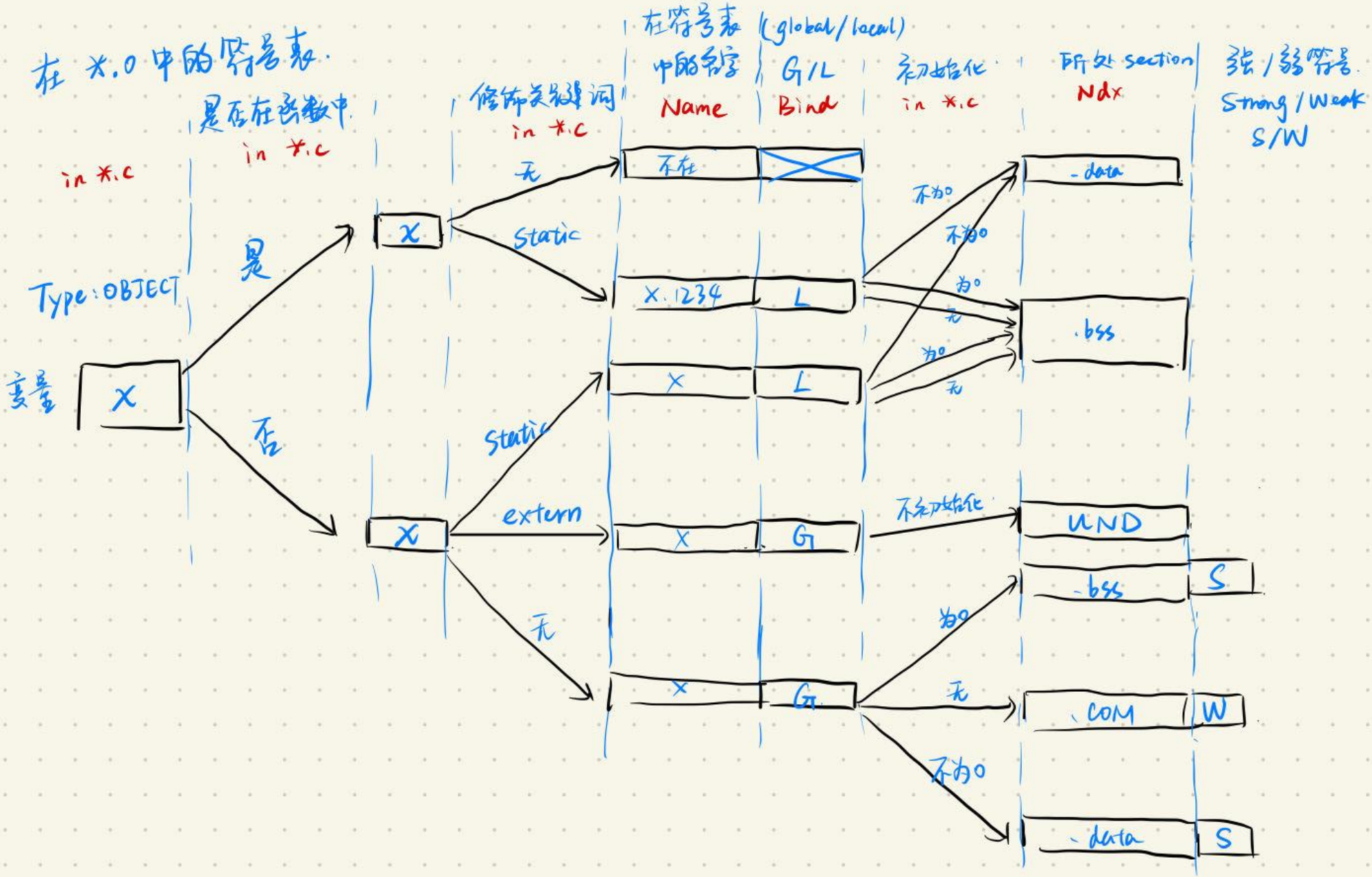
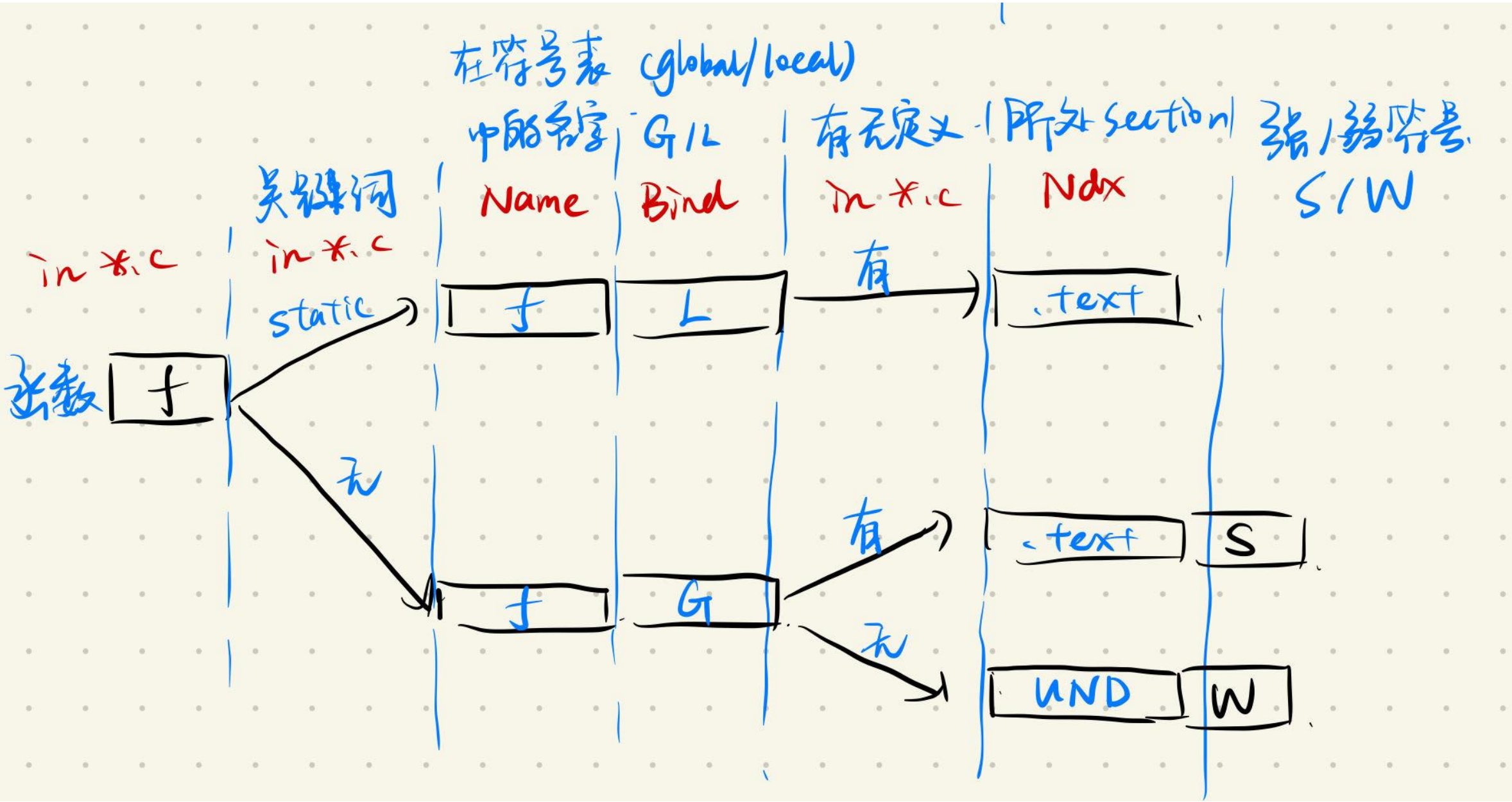
链接补充、作业讲评

孙英博


在 *.o 中的符号表
是否在函数中
in *.c
修饰关键词
in *.c
符号表中的名字
Name
(global/local)
G/L
Bind
初始化
in *.c
所处 section
Ndx
强/弱符号
Strong/Weak
S/W
in *.c
Type:OBJECT
变量
x
是
否
x
x
无
static
static
extern
无
不在
x.1234
x
x
x
L
L
G
G
不为0
不为0
为0
无
为0
无
不初始化
.data
.bss
UND
.bss
.COM
.data
S
W
S


在符号表 (global/local)
函数名字
Name
G/L
Bind
有无定义
in *.c
所处section
Ndx
强/弱符号
S/W
关联词
in *.c
in *.c
函数
f
static
无
f
L
有
.text
f
G
有
.text
S
无
UND
W

Part B. （15分）使用 gcc foo.c m.c 生成 a.out。 其节头部表部分信息如下。已知 main 和 foo 的汇编代码相邻，且 Ndx 和 Nr 都是指节索引。请补充空缺的内容。

Section Headers:

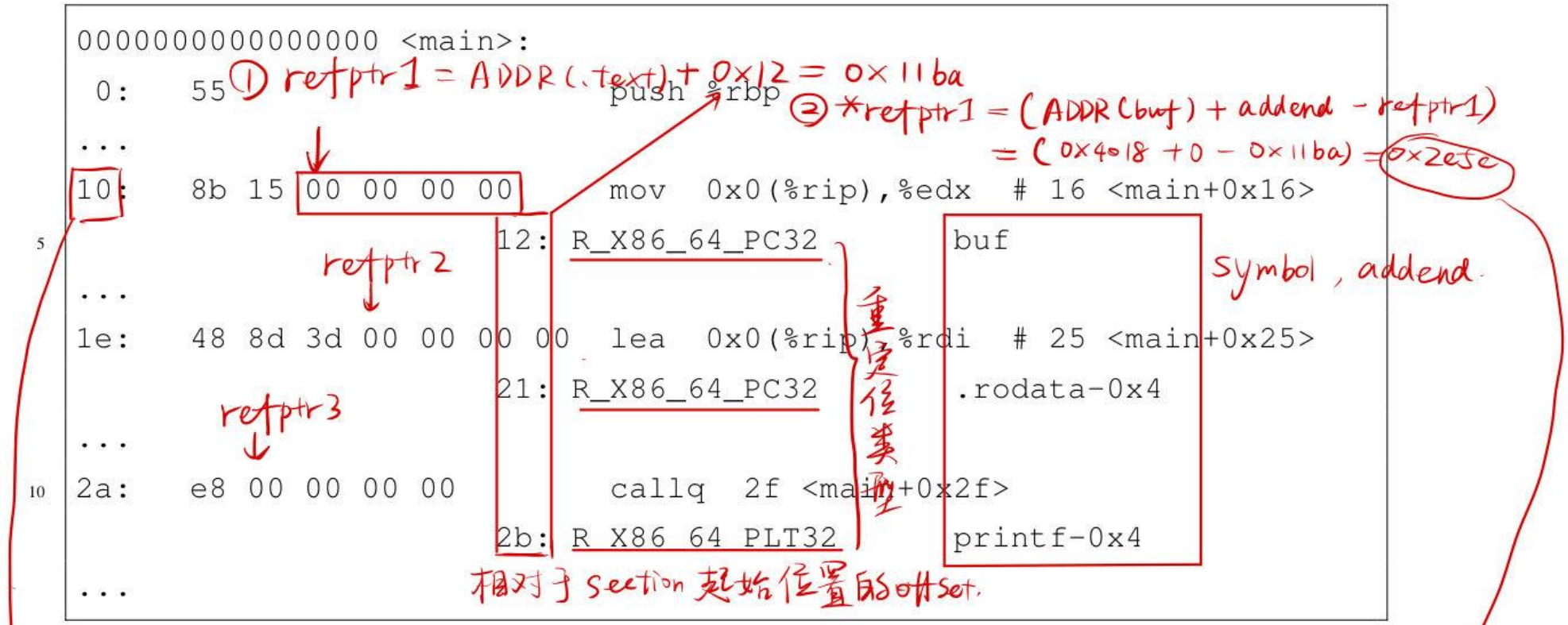
[Nr] Name	Type	Address	Offset	Size
[1] .interp	PROGBITS	00000000000002a8	000002a8	000000000000001c
[14] .text	PROGBITS	0000000000001050	00001050	0000000000000205
[16] .rodata	PROGBITS	0000000000002000	00002000	000000000000000a
[23] .data	PROGBITS	0000000000004000	00003000	0000000000000020
[24] .bss	NOBITS	0000000000004020	00003020	0000000000000010

Symbol Table:

Num:	Value	Size	Type	Bind	Ndx	Name
35:	0000000000004024	4	OBJECT	LOCAL	24	count.1797
54:	0000000000004010	8	OBJECT	GLOBAL	23	bufp0
59:	000000000000115a	78	FUNC	GLOBAL	14	foo
62:	0000000000004018	8	OBJECT	GLOBAL	23	buf
64:	00000000000011a8	54	FUNC	GLOBAL	14	main
68:	0000000000004028	8	OBJECT	GLOBAL	24	bufp1
51:	0000000000000000	0	FUNC	GLOBAL	UND	printf@@GLIBC_2.2.5

4. 接 2. 通过 objdump -dx foo.o 我们看到如下重定位信息。

```
0000000000000000 <main>:
 0:   55                      push   %rbp
...
10:   8b 15 00 00 00 00      mov    0x0(%rip),%edx        # 16 <main+0x16>
                             12: R_X86_64_PC32     buf
...
1e:   48 8d 3d 00 00 00 00   lea    0x0(%rip),%rdi        # 25 <main+0x25>
                             21: R_X86_64_PC32     .rodata-0x4
...
2a:   e8 00 00 00 00         callq  2f <main+0x2f>
                             2b: R_X86_64_PLT32    printf-0x4
...
```



假设链接器生成 a.out 时已经确定：foo.o 的.text 节在 a.out 中的起始地址为 ADDR(.text)=0x11a8。

请写出重定位后的对应于 main+0x10 位置的代码。

```
1168:   8b 15 5e 2e 00 00      mov   0x2e5e(%rip),%edx
```

而 main+0x1e 处的指令变成

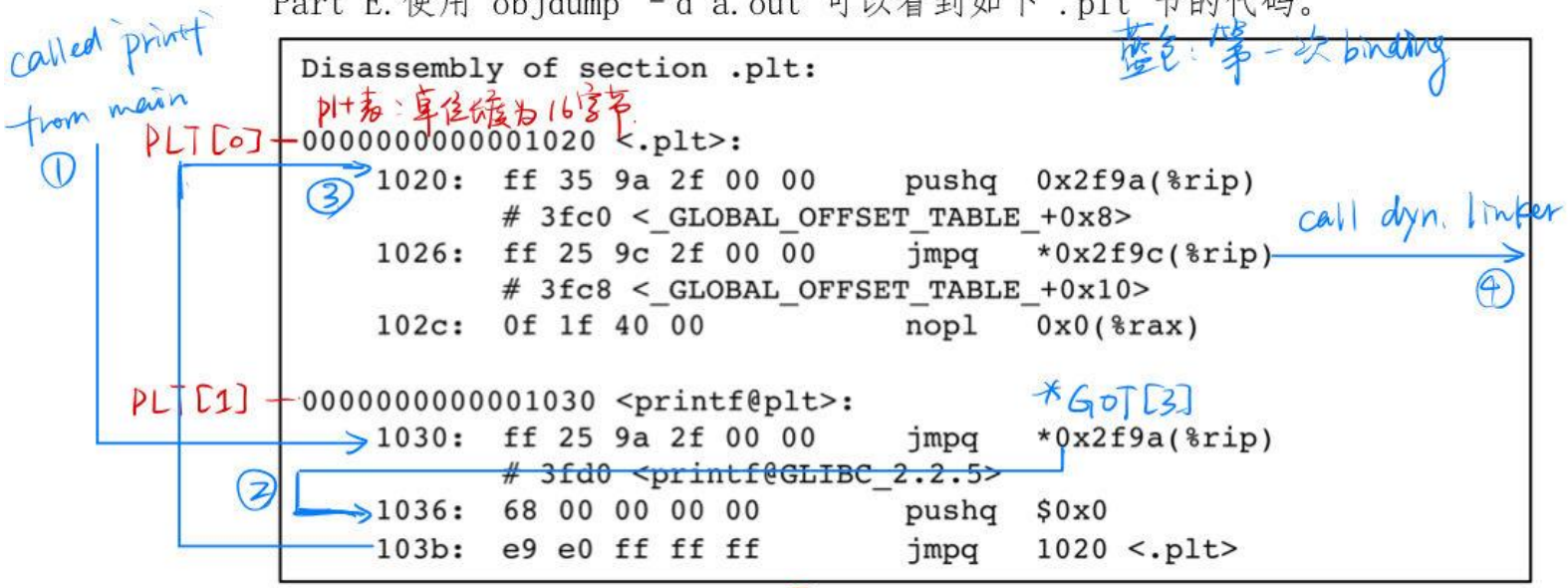
```
11c6: 48 8d 3d 37 0e 00 00     lea   0xe37(%rip),%rdi
```

△ 0x11c6 + 0x7 + 0xe37 = 2004

11c6+7:

可见字符串"%d %d" 在 a.out 中的起始地址是 0x 2004 。

Part E. 使用 objdump -d a.out 可以看到如下 .plt 节的代码。



a) 完成 main+0x2a 处的重定位。① retptr = 0x11a8 + 0x26 = 0x11d3

11d2 : e8 59 fe ff ff callq <printf@plt>

② *retptr = [Addr(printf) + addend - retptr] = [0x1030 - 0x4 - 0x11d3] = -0x1a7

b) printf 的 PLT 表条目是 PLT[1],GOT 表条目是 GOT[3](填写数字)。

GOT表：单位缩8字节

c) 使用 gdb 对 a.out 进行调试。某次运行时 main 的起始地址为 0x555555551a8,那么当加载器载入内存而尚未重定位 printf 地址前，printf 的 GOT 表项的内容是 0x 5555 5555 5036。

原因：ASLR

ASLR 前

0x 11a8 (main)

0x 1036 (GOT[3])

ASLR 后.

0x5555 5555 51a8

?

- 思考:
- GOT表的基址是?
- bind之后，GOT[3]的内容是?