```
int eat_peaches(int i) {
    if (i == 10) return 1;
    else return (eat_peaches(i + 1) + 1) * 2;
}
```

填充汇编代码（Linux X86-64）中缺失的内容：

```
eat_peaches:                                    i    in %rdi
.LFB0:
    cmpl        __空格1__, %edi
    __空格2__   .L8
    movl        $1, %eax
    ret
.L8:
    __空格3__    $8, %rsp
    addl        $1, %edi
    call        eat_peaches
    leal        __空格4__(%rax,%rax), %eax
    addq        $8, __空格5__
    ret
```

空格	值
__空格(1)__	$10
__空格(2)__	jne
__空格(3)__	subq
__空格(4)__	2
__空格(5)__	%rsp

Q1.2. bfloat16 （2分）

bfloat16 是由Google提出的一种半精度浮点数， exp 域为8位， frac 域为7位， sign 域为1位。除了位宽度差别外， bfloat16 的其它规格符合IEEE 754标准。

现定义一个C语言 union （联合体）数据类型，如下所示：

0 0000001 0000001

```
union {
    bfloat16 f;
    unsigned short s;
}
```

0 011 1111 1000 0001

现在为 f 赋予 bfloat16 所能表示的最接近于 1 且大于 1 的数，在X86（小端序）机器上运行时， s 的十六进制值为多少?

s = 0x 3f 81

Q2 Getter & Setter（10分）

过程调用以及返回的顺序在一般情况下都是"过程返回的顺序恰好与调用顺序相反"，但是我们可以利用汇编以及对运行栈的理解来编写汇编过程打破这一惯例。最典型的应用为有栈协程切换。

有如下汇编代码（**X86-32** 架构），其中 GET 过程唯一的输入参数是一个用于存储当前处理器以及栈信息的内存块地址（假设该内存块的空间足够大），而 SET 过程则用于恢复被 GET 过程所保存的处理器及栈信息，其唯一的输入参数也是该内存块地址。

```
GET:
movl 4(%esp), %eax   #(A)
...
movl %edi, 20(%eax)
movl %esi, 24(%eax)
movl %ebp, 28(%eax)
movl %ebx, 36(%eax)
movl %edx, 40(%eax)
movl %ecx, 44(%eax)
movl $1, 48(%eax)
movl (%esp), %ecx    #(B)
movl %ecx, 60(%eax)
leal 4(%esp), %ecx   #(C)
movl %ecx, 72(%eax)
movl 44(%eax), %ecx
movl $0, %eax
ret
```

```
SET:
movl 4(%esp), %eax
...
movl 20(%eax), %edi
movl 24(%eax), %esi
movl 28(%eax), %ebp
movl 36(%eax), %ebx
movl 40(%eax), %edx
movl 44(%eax), %ecx
movl 72(%eax), %esp    #(D)
pushl 60(%eax)         #(E)
movl 48(%eax), %eax
ret
```

在理解代码的基础上，回答下列问题:

1. SET 过程的返回地址是什么，其返回值是多少?

是GET的返回地址. 返回值是1

2. 代码段中的 (A) 指令执行后， eax 中存放的是什么?

~~是GET的返回地址~~、存储当前处理器以及栈信息的内存块地址.

(B) 指令执行后，ecx 中存放的是什么?

GET的返回地址.

(C) 指令的作用是什么?

获取调传入给GET的那个内存块地址，然后就可以在下一步存入指定内容中.

因为~~不能这样~~ 下一个操作数均为内存中的地址，所以需要
不允许 寄存器中转一下.

(E) 指令的作用是什么?

将GET的返回地值压入栈中，这样GET就可以返回到
GET的返回地址那了.

并将 (D) 指令补充完整。

72.

Q3 Fibonnaci求解（8分）

✓ **致谢**

感谢 计23·王浩然 同学、计25·张恒瑞 同学、计28·崔灏睿 同学、计21·李双宇 同学 抽空参与试做并提供及时反馈，协助优化本题的设计。

Fibonacci数列是一个非常经典的数列，其定义如下:

$$F(n) = F(n-1) + F(n-2)$$
$$F(0) = 0, F(1) = 1$$

为了巩固对X86-64汇编的掌握，我们提供了一个基于X86-64和Linux系统调用、 **求解Fibonacci数列中第n项** 的手写汇编代码，但是里面有一些bug，需要交给你来修复。 **(预期**

本道实践题的技术涉及了本课程多个知识点：

- Linux系统调用：通过X86-64提供的系统调用接口请求系统服务，参考 CSAPP 第8.1节；
- `as` 与 `ld` 命令：汇编代码的汇编、编译与链接，参考 CSAPP 第7章；
- Python的 `subprocess` 模块使用：使用 `subprocess` 创建子进程，并在其执行过程中与之交互，参考 CSAPP 第8.2 ~ 8.4节。

我们希望在未来的课上介绍这些知识点后，再配合实践题展开介绍这些技术。

Q4 实践题调研（5分）

实践题的引入是 本学期《计算机系统概论》课 课后作业的一次新尝试：通过一道道实践题为大家介绍一些与课上知识相关的编程trick，帮助大家一步步入门Linux环境编程。

在三次作业中我们先后布置了以下几道实践题：

☑ 作业一 - Q3.1 整数加法溢出检测
☑ 作业一 - Q3.2 字节序
☐ 作业二 - Q2.3 分支跳转指示
☑ 作业三 - Q3 Fibonacci求解

请勾选出截至目前 **你最喜欢的实践题（可多选）**，并从下面选项中挑选出 **你喜欢的理由**：

☑ 这道实践题帮助我很好的将课上知识应用于实践中
☑ 这道实践题有助于提升我的Linux编程技术
☑ 这道实践题获得答案的难度较低

在设计实践题时，为了降低大家的压力，我们给出了执行流程及结果供大家作答（本次实践题除外）。不过话说回来，挺好奇你有尝试过按照流程编译运行该代码吗？

☑ 尝试并成功复现结果
☐ 未尝试：没有想去尝试的欲望（太简单了，不复现也能做完）
☐ 未尝试：时间很忙，没有时间去coding
☐ 未尝试：没有配置环境 or 在配置的环境下复现时出现error
☐ 未尝试：看不懂题目 or 导引不够清晰

学有余力、有兴趣参与实践题试做的同学欢迎随时联系助教 yuxuanzh23@mails.tsinghua.edu.cn，邮件主题请注明【计系概实践题试做】。