作业三

本次作业一共分为4道大题，**总分30分**：

- Q1 是综合题：考察对课上内容的掌握；
- Q2 是分析题：具体来说你需要分析汇编代码的执行过程及其用途；
- Q3 为实践题：通过实际的X86汇编编程，让大家能更好的理解X86汇编；
- Q4 为多选题：与课调委沟通后，我们希望收集大家对作业的反馈，一同改进课程质量。

> ⚠ **打包提交本次作业**
>
> 具体来说你需要将 书面作业的答案 + 经 `check.py` 测试 的 `fib.s` 两个文件**打包上传**到网络学堂。
>
> 假如你的学号为 `2022011451`，则可以通过 `tar` 或 `zip` 命令打包：
>
> ```
> $ ls .  # 显示当前作业目录
> fib.s  作业三_答案.pdf/.docx/.md
> $ tar -czvf 2022011451.tar.gz *
> $ zip 2022011451.zip *
> ```
>
> 将打包好的 `2022011451.tar.gz` 或 `2022011451.zip` 上传到网络学堂。

Q1. 综合（7分）

Q1.1. 填充（5分）

编程解决猴子吃桃问题：每天吃一半再多吃一个，第十天想吃时候只剩一个，问总共有多少。
该程序的 C语言程序如下：

```
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:
.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)__`	
`__空格(2)__`	
`__空格(3)__`	
`__空格(4)__`	
`__空格(5)__`	

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：

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

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_____

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 _____(%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?

2. 代码段中的 `(A)` 指令执行后， `eax` 中存放的是什么?

`(B)` 指令执行后，`ecx` 中存放的是什么?

`(C)` 指令的作用是什么?

`(E)` 指令的作用是什么?

并将`(D)`指令补充完整。

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**

用时：< 8小时)

详见`fib.s`的`TODO:`以及对应的`hint`。具体来说，你需要：

1. 手动实现`print_int`函数缺失的`itoa_digits`（4分）

```
.type print_int, @function
print_int:
    # number to be printed in %rdi
    pushq %rbp
    movq  %rsp, %rbp
    movq %rdi, %rax

    # 先令换行符入栈，最后输出时换行
    movq $0xa, %rsi
    pushq %rsi
    movq $1, %rbx   # 记录当前待输出的字符数量，因为含'\n'所以初始为1

itoa_digits:
    # TODO: 将数字转换为字符串，并逆序保存在栈上，使得打印时次序正确
    # hint0: 从低位开始向高位处理，低位先入栈，高位先出栈
    # hint1: 使用cqto指令从32位扩展至64位，再用idivq指令获取 商（%rax）和 余数（%rdx）
    # hint2: 通过加上 $DIGIT_0 将余数转换为字符，并压入栈中保存，注意压入的是 X86-64寄存器
    # hint3: 更新待输出的字符总数（%rbx）中，这将作为print_digits的结束依据
```

提示：可以先注释掉`call fib`，通过调用`print_int`函数，观察是否能正确输出`scan_int`的结果。

2. 修正`fib.s`中的`fib`函数（4分）

```
.type fib, @function
fib:
    # TODO: 修正fib函数，使得能够正确计算fibonacci数列
    # hint0: 可以使用GDB打断点，进行指令粒度的调试
    # hint1: 检查栈帧的分配与回收是否正确
    # hint2: 检查函数调用前后寄存器是否被正确保存与恢复
    # hint3: 检查函数调用时的参数是否正确传递

    # n is in %rdi
    pushq %rbp
    movq  %rsp, %rbp
    movq %rdi, %rax
    cmpq $2, %rdi
    jl fib_end
    decq %rdi
    call fib
    movq %rax, %r8
    decq %rdi
    call fib
    addq %r8, %rax
fib_end:
    movq %rsp, %rbp
    popq %rbp
    ret
```

提示：你可以手写一个C程序，生成汇编代码后交叉对比，找出`fib`函数中的bug；善用`pushq` / `popq`指令能优化栈的使用体验。

使得`fib.s`最终能通过编译并正确执行（见下）。

```
$ as -g fib.s -o fib.o      # -g 启用调试
$ ld fib.o -o fib
$ ./fib
30
832040
```

> **提示**
>
> - 本次实践题需要你熟练掌握GDB的指令粒度调试，可参考网络学堂上发布的《实验导引1·Bomb Lab讲解》。
> - 本次作业涉及了除法运算，具体可参考 CSAPP 第3.5.5节，**这部分内容属于考试范围**。
> - `fib`采用最朴素的递归实现，不需要额外实现记忆化搜索或其它优化；想一想，《数据结构》课中 二叉树的后序遍历 是如何借助栈来实现的。

我们提供了一个Python脚本进行随机化测试，从而验证你的汇编代码是否实现正确。

确认`fib.s`与`check.py`位于同一文件夹下：

```
$ tree .
.
├── ...         # 其它文件
├── check.py
└── fib.s
```

预期的执行：

```
$ python check.py
Congratz! You've passed the test!
```

你需要修改`fib.s`，并通过正确性测试。最后将`fib.s`连同书面作业一并打包上传到网络学堂。

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】。