

计算机系统概论（2024 秋）作业 3

1. 猴子吃桃问题

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，问总共共有多少。该程序的 C 语言程序如下，请在其对应汇编代码（Linux x86-64）内填入缺失内容。

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

汇编代码如下：

```
eat_peaches:  
    cmpl    $10, _____  
    je      _____  
    subq    $8, %rsp  
    addl    $1, %edi  
    call    eat_peaches  
    leal    2(%rax, _____), %eax  
    jmp     _____  
.L3:  
    movl    $1, %eax  
    _____  
.L2:  
    addq    _____, %rsp  
    ret
```

2. 栈变量位置

有下列 C 代码以及对应的汇编代码 (Linux x86-64), 请填充下表, 即给出各个变量或者寄存器在栈中的存储位置 (以相对于栈帧基址寄存器%rbp 的十进制偏移量形式给出, 可正可负); 如果无法以“在栈中的存储位置”形式给出, 请说明理由。

.LC0:

.string "a[0] = 0x%x, a[1] = 0x%x, buf = %s\n"

foo:

```
pushq    %rbp
movq     %rsp, %rbp
pushq    %rbx
subq     $24, %rsp
movl     %edi, %ebx
leaq     -32(%rbp), %rdi
movl     $0, %eax
call     gets
leaq     -32(%rbp), %rcx
movl     %ebx, %edx
movl     $-252579085, %esi
movl     $.LC0, %edi
movl     $0, %eax
call     printf
addq     $24, %rsp
popq     %rbx
popq     %rbp
ret
```

```
void foo(int x)
{
    int a[3];
    char buf[4];
    a[0] = 0xF0F1F2F3;
    a[1] = x;
    gets(buf);
    printf("a[0] = 0x%x, a[1] = 0x%x, buf = %s\n",
        a[0], a[1], buf);
}
```

变量	十进制形式的 offset 或者说明
a	
a[2]	
x	
buf	
buf[3]	
%rbx 的保存值	

3. 非常规过程调用

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。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。在理解代码的基础上，回答下列问题：

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

- (1) SET 过程的返回地址是什么，其返回值是多少？
- (2) 完成以下子问题。（注：写出这些指令对 GET/SET 正确运行的作用，不要用形如“xxx 指令的作用是把%eax 的值赋值给%ecx”，来解释这条指令的含义）
 1. (A) 指令执行后，eax 中存放的是？
 2. (B) 指令执行后，ecx 中存放的是什么？
 3. (C) 指令的作用是什么？
 4. 将 (D) 指令补充完整。
 5. (E) 指令的作用是什么？

4. 类函数调用

根据以下代码及汇编（-O0 生成）回答如下问题：

- (1) Test 类构造函数内的 this 指针是如何传递的？在 main 函数栈帧内的什么位置？
- (2) 类成员 fun 函数执行时是如何定位各个类成员变量的？

提示：本题的汇编中出现了复杂函数名，这涉及到了 C++ 的名字修改 (name mangling) 机制，感兴趣的同学可以自行检索学习。特别地，_ZN4TestC2Eiiii 对应于 Test 类构造函数，后面的 5 个 i，指代 5 个 int 类型输入。

```

class Test{
public:
    int a;
    int b;
    int c;
    int d;
    int e;
    Test(int a, int b, int c, int d, int e) {
        this->a = a;
        this->b = b;
        this->c = c;
        this->d = d;
        this->e = e;
    }
    int fun() {
        int a1 = a;
        int b1 = b;
        int c1 = c;
        int d1 = d;
        int e1 = e;
        return 100;
    }
};

int main()
{
    Test test(1, 2, 3, 4, 5);
    test.fun();
    return 0;
}

```

```

__ZN4TestC2Eiiii:
    pushq    %rbp
    movq     %rsp, %rbp
    movq     %rdi, -8(%rbp)
    movl     %esi, -12(%rbp)
    movl     %edx, -16(%rbp)

```

```

movl    %ecx , -20(%rbp)
movl    %r8d , -24(%rbp)
movl    %r9d , -28(%rbp)
movq    -8(%rbp) , %rax
movl    -12(%rbp) , %edx
movl    %edx , (%rax)
movq    -8(%rbp) , %rax
movl    -16(%rbp) , %edx
movl    %edx , 4(%rax)
movq    -8(%rbp) , %rax
movl    -20(%rbp) , %edx
movl    %edx , 8(%rax)
movq    -8(%rbp) , %rax
movl    -24(%rbp) , %edx
movl    %edx , 12(%rax)
movq    -8(%rbp) , %rax
movl    -28(%rbp) , %edx
movl    %edx , 16(%rax)
nop
popq    %rbp
ret

```

_ZN4Test3funEv :

```

pushq   %rbp
movq    %rsp , %rbp
movq    %rdi , -40(%rbp)
movq    -40(%rbp) , %rax
movl    (%rax) , %eax
movl    %eax , -4(%rbp)
movq    -40(%rbp) , %rax
movl    4(%rax) , %eax
movl    %eax , -8(%rbp)
movq    -40(%rbp) , %rax
movl    8(%rax) , %eax
movl    %eax , -12(%rbp)
movq    -40(%rbp) , %rax
movl    12(%rax) , %eax

```

	movl	%eax , -16(%rbp)
	movq	-40(%rbp) , %rax
	movl	16(%rax) , %eax
	movl	%eax , -20(%rbp)
	movl	\$100 , %eax
	popq	%rbp
	ret	
main :		
	pushq	%rbp
	movq	%rsp , %rbp
	subq	\$32 , %rsp
	leaq	-32(%rbp) , %rax
	movl	\$5 , %r9d
	movl	\$4 , %r8d
	movl	\$3 , %ecx
	movl	\$2 , %edx
	movl	\$1 , %esi
	movq	%rax , %rdi
	call	__ZN4TestC1Eiiii
	leaq	-32(%rbp) , %rax
	movq	%rax , %rdi
	call	__ZN4Test3funEv
	movl	\$0 , %eax
	leave	
	ret	

5. 补全 C 代码

有如下三类结构联合定义，请根据左侧的汇编语言 (x86-32)，补齐右侧的 C 语言。
补全的结构：

<pre> struct s1 { char a[3]; union u1 b; int c; }; </pre>	<pre> struct s2 { struct s1 *d; char e; int f[4]; struct s2 *g; }; </pre>	<pre> union u1 { struct s1 *h; struct s2 *i; char j; }; </pre>
---	---	--

- A. proc1:
- ```

pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%eax
movl 12(%eax),%eax
movl %ebp,%esp
popl %ebp
ret

```
- int proc1(struct s2 \*x)
- ```

{
    return x->_____ ;
}

```
- B. proc2:
- ```

pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%eax
movl 4(%eax),%eax
movl 20(%eax),%eax
movl %ebp,%esp
popl %ebp
ret

```
- int proc2(struct s1 \*x)
- ```

{
    return x->_____ ;
}

```
- C. proc3:
- ```

pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%eax
movl (%eax),%eax
movsbl 4(%eax),%eax
movl %ebp,%esp
popl %ebp
ret

```
- char proc3(union u1 \*x)
- ```

{
    return x->_____ ;
}

```
- D. proc4:
- ```

pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%eax
movl (%eax),%eax
movl 24(%eax),%eax
movl (%eax),%eax
movsbl 1(%eax),%eax
movl %ebp,%esp
popl %ebp
ret

```
- char proc4(union u1 \*x)
- ```

{
    return x->_____ ;
}

```

6. 内存布局分析

已知以下 C++ 代码与对应的运行结果，对源代码进行补全并绘制 struct A 各变量在内存中的存放位置。

```

#include <iostream>
#include <cstdlib>
using namespace std;
struct A {
    T1 a;
    T2 b;
    struct B {
        T3 c;
        T4 d;
    };
    B e[N];
};
int main() {
    A s;
    size_t size_A = sizeof(s);
    std::cout << "size of A:" << size_A << std::endl;
    unsigned char* p = (unsigned char*) &s;
    for (int i = 0; i < size_A; i++) p[i] = 0xaa;
    std::cout << "a:" << s.a << std::endl;
    std::cout << "b:" << s.b << std::endl;
    std::cout << "e[0].c:" << s.e[0].c << std::endl;
    std::cout << "e[0].d:" << s.e[0].d << std::endl;
    return 0;
}

```

运行结果如下：

```

size of A: 96
a: -21846
b: 12297829382473034410
e[0].c: -1431655766
e[0].d: -3.72066e-103

```

请补全以下类型和常数 T1: _____ T2: _____ T3: _____ -
 _____ T4: _____ N: _____, 并回答: struct A 的内存布局 (需
 绘制出 struct B 中各变量)。