

## 计算机系统概论（2023 秋） 作业 3

1. 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，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;  
    }  
}
```

```
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. 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. 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：

<pre>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</pre>	<pre>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</pre>
---	---

- 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。（注：写出这些指令对 GET/SET 正确运行的作用，不要形如“xxx 指令的作用是把 \$eax 的值赋值给 \$ecx”的解释这条指令的含义）

4. 请分别对照下列的 C 代码与对应的汇编代码，解释 C 函数是如何传入 struct 类型参数的，并画出 call input\_struct 时栈的 layout，辅以说明来解释。

1) gcc -Og ...

```
typedef struct{
    int age; int bye; int coo; int ddd; int eee;
} TEST_Struct;
int i = 2;
int input_struct(TEST_Struct in_struct) {
    return in_struct.eee + in_struct.age*2 ;
}
int function2() {
    TEST_Struct main_struct;
    main_struct.age = i;
    main_struct.bye = i;
    main_struct.coo = 2*i;
    main_struct.ddd = i;
    main_struct.eee = i;
    return input_struct(main_struct);
}
```

input\_struct:

```
movl    8(%rsp), %eax    #age
addl    %eax, %eax
addl    24(%rsp), %eax   #eee
ret
```

function2:

```
subq    $56, %rsp
movl    i(%rip), %eax
movl    %eax, 24(%rsp)   #age
movl    %eax, 28(%rsp)   #bye
leal    (%rax,%rax), %edx
movl    %edx, 32(%rsp)   #coo
movl    %eax, 36(%rsp)   #ddd
movq    24(%rsp), %rdx
movq    %rdx, (%rsp)     #age/bye
movq    32(%rsp), %rdx
movq    %rdx, 8(%rsp)    #coo/ddd
movl    %eax, 16(%rsp)   #eee
call    input_struct
addq    $56, %rsp
ret
```

- 2) C 代码不变，通过 `gcc -O1/2 ...` 编译后的汇编如下：

```
input_struct:
    movl    24(%rsp), %eax
    movl    8(%rsp), %edx
    leal    (%rax,%rdx,2), %eax
    ret

function2:
    movl    i(%rip), %eax
    leal    (%rax,%rax,2), %eax
    ret
```

请分析针对这段代码，编译器做了什么优化工作。

- 3) 如果在上面的 C 代码的 `int input_struct (...)` 声明前加上 `static`，`gcc -O1/2 ...` 编译后的代码如下：

```
function2:
    movl    i(%rip), %eax
    leal    (%rax,%rax,2), %eax
    ret
```

请分析针对这段代码，编译器做了什么优化工作。

5. 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。

<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. 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：

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: \_\_\_\_\_

struct A 的内存布局（需绘制出 struct B 中各变量）: