计算机系统概论（2022 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, %edi
        je      .L3
        subq    $8, %rsp
        addl    $1, %edi
        call    eat_peaches
        leal    2(%rax, %rax), %eax
        jmp     .L2
.L3:
        movl    $1, %eax    ret

.L2:
        addq    $8, %rsp
        ret
```

扫描全能王 创建

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：

```
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
```

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。

(1) GET 函数的返回地址；1

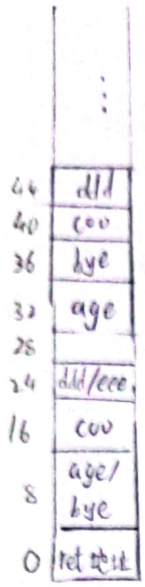
(2) A：该内存块地址

B：GET的返回地址

C：将 4(%esp) 地址保存进 72(%eax)，便于GET将%rsp的地址恢复

E：将返回地址压入，使 SET 结束后能直接返回 GET前的位置

扫描全能王 创建



调用 input_struct 时先将返回地址压入，再取 8(%rsp) 存放的 age 的值放入 %eax，再将 2 倍的 %eax 加上 24(%rsp) 存放的 eee 的值

2) C 代码不变，通过 gcc -01/2 …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。

input_struct ：与(1)区别不大，用了 %edx 寄存器

function ：没有调用 input_struct，直接计算

3) 如果在上面的 C 代码的 int input_struct（…）声明前加上 static，gcc -01/2 … 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。

因 static 函数仅在本文件中使用，故其永远不会用到，故直接在编译中省略

扫描全能王 创建

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。

```
struct s1 {
  char a[3];
  union u1 b;
  int c;
};
```

```
struct s2 {
  struct s1 *d;
  char e;
  int f[4];
  struct s2 *g;
};
```

```
union u1 {
  struct s1 *h;
  struct s2 *i;
  char j;
};
```

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->_____d[1]_____;
}
```

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->_____b.i->d[3]_____;
}
```

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->_____i->e_____;
}
```

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->_____i->g->d->a[1];
}
```

扫描全能王 创建

请补全以下类型和常数

T1: short T2: unsigned long long T3: int T4: double

N: 5

struct A 的内存布局 （需绘制出 struct B 中各变量）：

a	–	b	e[0].c	–	e[0].d	e[1].c	–	···	e[4].d
0-2	2-8	8-16	16-20	20-24	24-32	32-36	36-40		88-96

扫描全能王 创建

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	存入 %esi
a[2]	未赋值
x	仅用 %ebx 保存
buf	-32
buf[3]	-29
%rbx 的保存值	-8

扫描全能王 创建