北京航空航天大学 2013－2014 学年 第一学期 《计算机组成与体系结构》期末考试

Class number班级\_\_\_\_\_\_\_\_\_\_ Student ID学号 \_\_\_\_\_\_\_\_\_\_\_

Student Name姓名\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Score成绩 \_\_\_\_\_\_\_\_\_\_\_\_

答题页1 （答案全部填写在答题页中，其它地方无效）

Problem 1.（20 points）

1\_\_\_\_ 2\_\_\_\_ 3\_\_\_\_ 4\_\_\_\_ 5\_\_\_\_ 6\_\_\_\_ 7\_\_\_\_ 8\_\_\_\_ 9\_\_\_\_ 10\_\_\_\_\_ 11\_\_\_\_\_ 12\_\_\_\_\_ 13\_\_\_\_\_ 14\_\_\_\_\_ 15\_\_\_\_\_ 16\_\_\_\_\_ 17\_\_\_\_\_ 18\_\_\_\_\_ 19\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| Function | Intel IA32 | Intel x86 64 |
| sizeof (char) |  |  |
| sizeof (int) |  |  |
| sizeof (void\*) |  |  |
| sizeof (long) |  |  |

20

Problem 2.（10 points）

|  |  |  |
| --- | --- | --- |
| Value | Floating Point Bits | Rounded value |
| 9/32 | 001 00 | 1/4 |
| 1 |  |  |
| 12 |  |  |
| 11 |  |  |
| 1/8 |  |  |
| 7/32 |  |  |

Problem 3.（8 points）

|  |  |
| --- | --- |
| Code Block | Function Name |
| A |  |
| B |  |
| C |  |
| D |  |

Problem 4.（11 points）

Cases \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ should have "break".

0x400590: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 0x400598: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x4005a0: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 0x4005a8: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x4005b0: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 0x4005b8: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x4005c0: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 0x4005c8: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x4005d0: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 0x4005d8: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem 5.（12 points）

A: x = \_\_\_\_\_\_

B: string "0123456" is stored at \_\_\_\_\_\_\_\_\_\_\_

C: buf[0] = 0x\_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_

buf[1] = 0x\_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_

buf[2] = 0x\_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_

buf[3] = 0x\_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_

buf[4] = 0x\_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_

答题页2

D: Value at %ebp is \_\_\_\_\_\_\_\_\_\_\_

E: Value at %esp is \_\_\_\_\_\_\_\_\_\_\_

F:

Problem 6.（9 points）

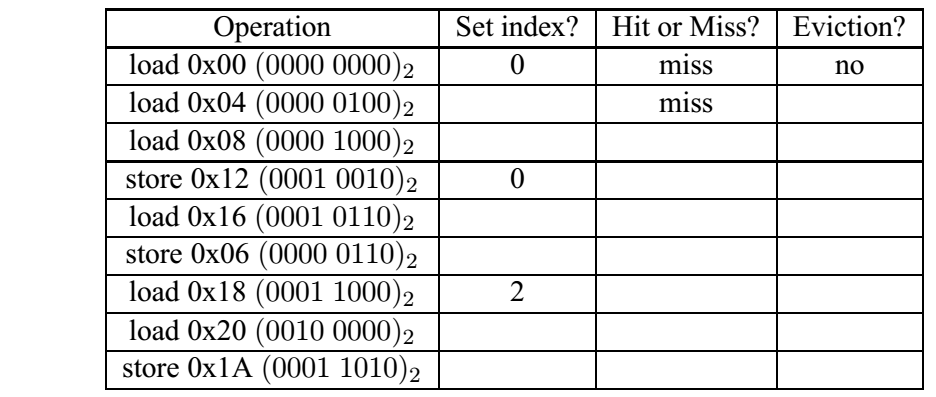
Answer: a=\_\_\_\_\_, b=\_\_\_\_\_, c=\_\_\_\_\_

Problem 7.（10 points）

(a). Cache size is \_\_\_\_\_ bytes

(b). Tag is \_\_\_\_\_ bits

(c).



Problem 8.（10 points）

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem 9.（10 points）

1. a. Physical address of PDE: \_\_\_\_\_\_\_\_\_\_\_

b. Physical address of PTE: \_\_\_\_\_\_\_\_\_\_\_

c. Success: The physical address accessed is \_\_\_\_\_\_\_\_\_\_\_

or

Failure: Address of table entry causing failure is \_\_\_\_\_\_\_\_\_\_\_

2. a. Physical address of PDE: \_\_\_\_\_\_\_\_\_\_\_

b. Physical address of PTE: \_\_\_\_\_\_\_\_\_\_\_

c. Success: The physical address accessed is \_\_\_\_\_\_\_\_\_\_\_

or

Failure: Address of table entry causing failure is \_\_\_\_\_\_\_\_\_\_\_

### Problem 1.

1. Consider the following code, what is the output of the printf? 下面代码的输出是什么？

int x = 0x15213F10 >> 4;

char y = (char) x;

unsigned char z = (unsigned char) x;

printf("%d, %u", y, z);

(a) -241, 15

(b) -15, 241

(c) -241, 241

(d) -15, 15

b. 0x15213F10 >> 4 为 0x015213F1，转为有无符号char类型后均为最低1字节 0xF1，

%d作为有符号数输出，为-15；%u作为无符号数，为241

2. In two’s compliment, what is −TMin? 补码中，−TMin的值是多少？

(a) TMin

(b) TMax

(c) 0

(d) −1

a. 补码不对称，−TMin=TMin

3. Let int x = −31/8 and int y = −31 >> 3. What are the values of x and y?

(a) x = −3, y = −3

(b) x = −4, y = −4

(c) x = −3, y = −4

(d) x = −4, y = −3

c. 除法向0舍入；右移向下舍入

4. In C, the expression ”15213U > −1” evaluates to:

(a) True (1)

(b) False (0)

b. 有无符号数混合运算时，所有数都被当做无符号数处理。-1的无符号数是很大的正数

5. In two’s compliment, what is the minimum number of bits needed to represent the number -1 and the number 1 respectively? 补码中，表示数字 -1和1需要的最小位数分别是多少？

(a) 1 and 2

(b) 2 and 2

(c) 2 and 1

(d) 1 and 1

a. 补码第一位为“1”则为负数，因此-1只需1位，要表示正数1则需要两位“01”

6. Consider the following program. Assuming the user correctly types an integer into stdin, what will the program output in the end? 假设用户正确输入了一个整数，下面代码的输出是什么？

#include <stdio.h>

int main() {

int x = 0;

printf ("Please input an integer:");

scanf ("%d",x);

printf ("%d", (!!x)<<31);

}

(a) 0

(b) T Min

(c) Depends on the integer read from stdin 取决于用户输入的整数

(d) Segmentation fault 段错误

d. scanf函数第二个参数前缺少&符号，程序运行报错

7. Which of the following registers stores the return value of functions in Intel x86 64?

Intel x86 64系统中哪个寄存器保存函数的返回值？

(a) %rax

(b) %rcx

(c) %rdx

(d) %rip

(e) %cr3

a. 寄存器的规定用法

8. The leave instruction is effectively the same as which of the following: Leave指令相当于：

(a) mov %ebp, %esp

pop %ebp

(b) pop %eip

(c) mov %esp, %ebp

pop %esp

(d) ret

a. leave指令的作用：恢复原ebp

9. Select the two’s complement negation of the following binary value: 0000101101:

二进制数0000101101负值的补码是：

(a) 1111010011

(b) 1111010010

(c) 1000101101

(d) 1111011011

a. 负值补码为正值所有位取反再在最后一位加1

10. Which line of C-code will perform the same operation as leal 0x10(%rax,%rcx,4),%rax?

下面哪行c代码和 leal 0x10(%rax,%rcx,4),%rax 操作相同？

(a) rax = 16 + rax + 4 \* rcx

(b) rax = \*(16 + rax + 4 \* rcx)

(c) rax = 16 + \* (rax + 4 \* rcx)

(d) \*(16 + rcx + 4 \* rax) = rax

(e) rax = 16 + 4 \* rax + rcx

a. leal直接计算源地址，存入目的寄存器

11. Which of the following assembly instructions is invalid in Intel IA32 Assembly?

IA32系统中，下面哪条汇编指令是非法的？

(a) pop %eip

(b) pop %ebp

(c) mov (%esp),%ebp

(d) lea 0x10(%esp),%ebp

a. 指令寄存器eip不可修改

12. For the Unix linker, which of the following accurately describes the difference between global symbols and local symbols?

对UNIX链接器，下面哪个说法准确描述了全局符号和本地符号的差异？

1. There is no functional difference as to how they can be used or how they are declared.

它们的使用和声明没有功能上的区别。

1. Global symbols can be referenced by other modules (files), but local symbols can only be referenced by the module that defines them.

全局符号可以被其它模块引用，而本地符号只能被定义它们的模块引用。

1. Global symbols refer to variables that are stored in .data or .bss, while local symbols refer to variables that are stored on the stack.

全局符号指的是保存在.data或.bss中的变量，而本地符号指的是保存在栈里的变量。

1. Both global and local symbols can be accessed from external modules, but local symbols are declared with the “static” keyword.

全局符号和本地符号都可以被外部模块访问，但本地符号用“static”关键字声明。

b. 全局符号和本地符号概念

1. Which of the following is true concerning dynamic memory allocation?

关于动态存储器分配，下列哪项是正确的？

1. External fragmentation is caused by chunks which are marked as allocated but actually cannot being used.

外部碎片是由标记为已分配但实际上不能被使用的片引起的。

1. Internal fragmentation is caused by padding for alignment purposes and by overhead to maintain the heap data structure (such as headers and footers).

内部碎片是由为了对齐目的的填充和保持堆数据结构（如头部和脚部）的开销引起的。

1. Coalescing while traversing the list during calls to malloc is known as immediate coalescing.

调用malloc过程中遍历链表时的合并是立即合并。

1. Garbage collection, employed by calloc, refers to the practice of zeroing memory before use.

calloc采用的垃圾收集是指存储器使用前归零。

b. 内部和外部碎片概念

For the next 3 questions, consider the following code running on a 32-bit Linux system.

下面3个问题，考虑运行在32位linux系统上的代码

int main()

{

long a, \*b, c;

char \*\*p;

p = calloc(8, sizeof(char)); /\*calloc returns 0x1dce1000\*/

a = (long) (p + 0x100);

b = (long\*) (\*p + 0x200);

c = (int) (b + 0x300);

printf("p=%p a=%x b=%p c=%x\n", p, a, b, c);

exit(0);

}

考察指针加的概念。

14. When printf is called, what is the hex value of variable a? printf被调用时，变量a的值是？

(a) Can’t tell

(b) 0x1dce1100

(c) 0x1dce1400

(d) 0x1dce1800

c. p为char\*的指针，每加1，地址增加4.

0x1dce1000 + 0x100 \* 4 = 0x1dce1400

15. When printf is called, what is the hex value of variable b? printf被调用时，变量b的值是？

(a) Can’t tell

(b) 0x1dce1200

(c) 0x1dce2000

(d) 0x200

(e) 0x800

d. \*p为char的指针，每加1，地址增加1.

\*p = 0, b = \*p + 0x200 = 0x200

16. When printf is called, what is the hex value of variable c? printf被调用时，变量c的值是？

(a) Can’t tell

(b) 0x1dce2a00

(c) 0x1dce4400

(d) 0xc00

(e) 0xe00

e. b为long的指针，每加1，地址增加4.

b + 0x300 = 0x200 + 0x300 \* 4 = 0xe00

17. Which of the following is not a default action for any signal type?

下面哪项不是对某个信号类型的默认行为？

(a) The process terminates. 进程终止。

(b) The process reaps the zombies in the waitlist. 进程回收等待列表中的僵死进程。

(c) The process stops until restarted by a SIGCONT signal.

进程停止直到被SIGCONT信号重启。

(d) The process ignores the signal. 进程忽略信号。

(e) The process terminates and dumps core. 进程终止并转储存储器。

b. 信号处理概念。

18. A system uses a two-way set-associative cache with 16 sets and 64-byte blocks. Which set does the byte with the address 0xdeadbeef map to?

一个系统使用2路组相联高速缓存，有16组和64字节的块。地址0xdeadbeef映射到哪组？

(a) Set 7

(b) Set 11

(c) Set 13

(d) Set 14

b. 有16组，s = 4；64字节的块，b = 6。地址中，末6位（0~5位）为块偏移，中间4位（6~9位）为组索引。e = 1110，6~9位为 1011，即11.

19. When it suceeds, longjmp is called once and returns how many times?

调用成功时，longjmp被调用一次，返回多少次？

(a) 0

(b) 1

(c) 2

(d) 3

a. longjmp 从不返回。

20. Please fill in the return value for the following function calls on both an Intel IA32 and Intel x86 64 system.

在答题页1的表中填写函数调用在Intel IA32和Intel x86 64系统中的返回值。

|  |  |  |
| --- | --- | --- |
| Function | Intel IA32 | Intel x86 64 |
| sizeof (char) | 1 | 1 |
| sizeof (int) | 4 | 4 |
| sizeof (void\*) | 4 | 8 |
| sizeof (long) | 4 | 8 |

### Problem 2.

Consider the following 5-bit floating point representation based on the IEEE floating point format. This format does not have a sign bit – it can only represent nonnegative numbers.

考虑下面基于IEEE浮点格式的5位浮点表示，没有符号位，只能表示非负数。

• There are k = 3 exponent bits. The exponent bias is 3.

有k = 3个阶码位，偏置值是3

e不是全零或全1时为规格化数；规格化数e的取值范围: 1~6，E的取值范围：-2~3，超出此范围则用非规格化数编码。

• There are n = 2 fraction bits.

有n = 2个小数位

Below, you are given some decimal values, and your task is to encode them in floating point format. In addition, you should give the rounded value of the encoded floating point number. To get credit, you must give these as whole numbers (e.g., 17) or as fractions in reduced form (e.g., 3/4). Any rounding of the significand is based on *round-to-even*.

下面给出了一些十进制数值，你的任务是将它们编码为浮点格式。另外，给出被编码的浮点数舍入后的值。给出整数（例如17）或者分数（例如3/4）。使用舍入到偶数的原则。

|  |  |  |
| --- | --- | --- |
| Value | Floating Point Bits | Rounded value |
| 9/32 | 001 00 | 1/4 |
| 1 | 011 00 | 1 |
| 12 | 110 10 | 12 |
| 11 | 110 10 | 12 |
| 1/8 | 000 10 | 1/8 |
| 7/32 | 001 00 | 1/4 |

非规格化数最大为：000 11, 即3/4 \* 2-2 = 3/16 （6/32）

规格化数最小为：001 00, 即1.00 \* 2-2 = 1/4 （8/32）

规格化数最大为：110 11, 即1.11 \* 23 = 1110 = 14

前3个数在规格化数范围内: 1.00\*20 1.10\*23 1.011\*23

1/8在非规格化数范围内: 0.10\*2-2

7/32介于两者之间，先写成小数进行舍入: 0.00111

1: f=0.00, 小数位00; e-3=0, e=3, 阶码位011; 浮点编码为011 00, 无舍入，rv=1

12: f=0.10, 小数位10; e-3=3, e=6, 阶码位110; 浮点编码为110 10, 无舍入，rv=12

11: f=0.011, 舍入到偶数，f=0.10, 小数位10; e-3=3, e=6, 阶码位110; 浮点编码为

110 10, 有舍入，舍入后值为1100, rv=12

1/8: 在非规格化数范围内

e=0, E=1-3=-2; f=0.10, 小数位10; 无舍入，rv=1/8

7/32: 介于非规格化数和规格化数取值范围之间，需舍入。向下舍入为0.0011，

向上舍入为0.0100；根据舍入到偶数原则，应向上舍入,1.00\*2-2，为最小的规格化数

001 00，rv=1/4

### Problem 3.

Consider the following data structure declaration:

struct ms\_pacman{

short wire;

int resistor;

union transistor{

char bjt;

int \*mosfet;

long vacuum\_tube[2];

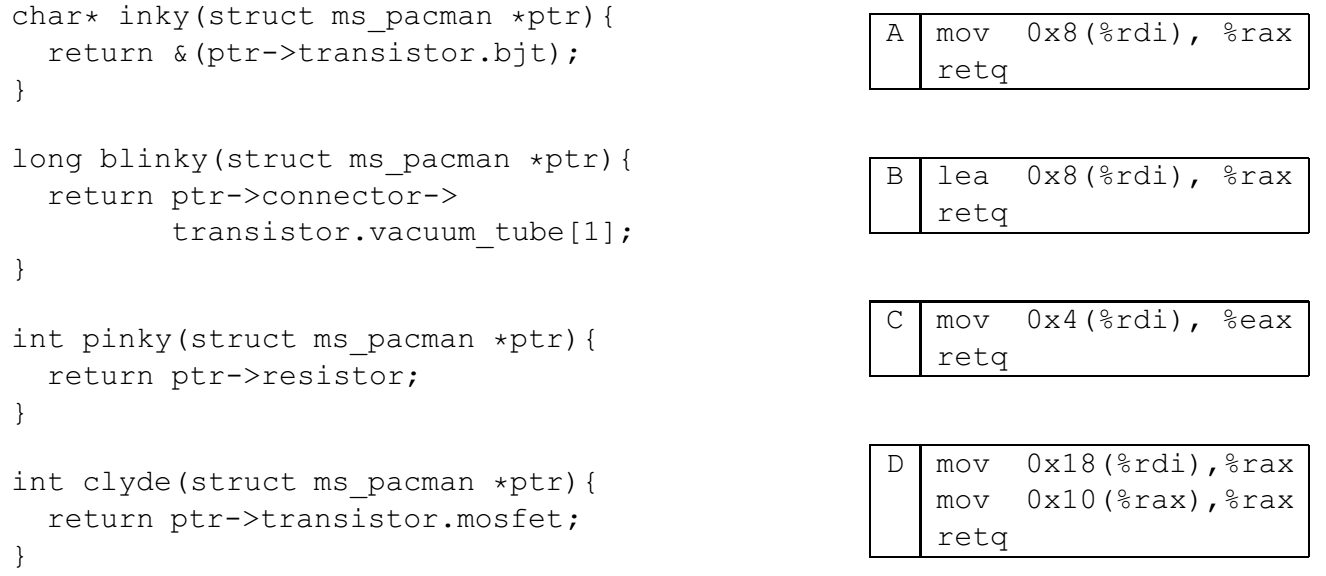
} transistor;

struct ms\_pacman \*connector;

};

Below are given four C functions and four x86-64 code blocks.

下面给出4个c函数和4段x86-64代码。



In the following table, next to the name of each x86-64 code block, write the name of the C function that it implements.

下表中，在x86-64代码块名字的右边写出对应c函数名。

|  |  |
| --- | --- |
| Code Block | Function Name |
| A | clyde |
| B | inky |
| C | pinky |
| D | blinky |

根据字节对齐要求（resistor 4字节对齐），wire 4字节（实际用2，偏移0），resistor 4字节（偏移4），transistor 16字节（偏移8，vacuum\_tube[0]偏移8，vacuum\_tube[1]偏移16），connector 8字节（偏移24），struct共32字节。

inky: 返回值为起始地址+8字节，对应代码B

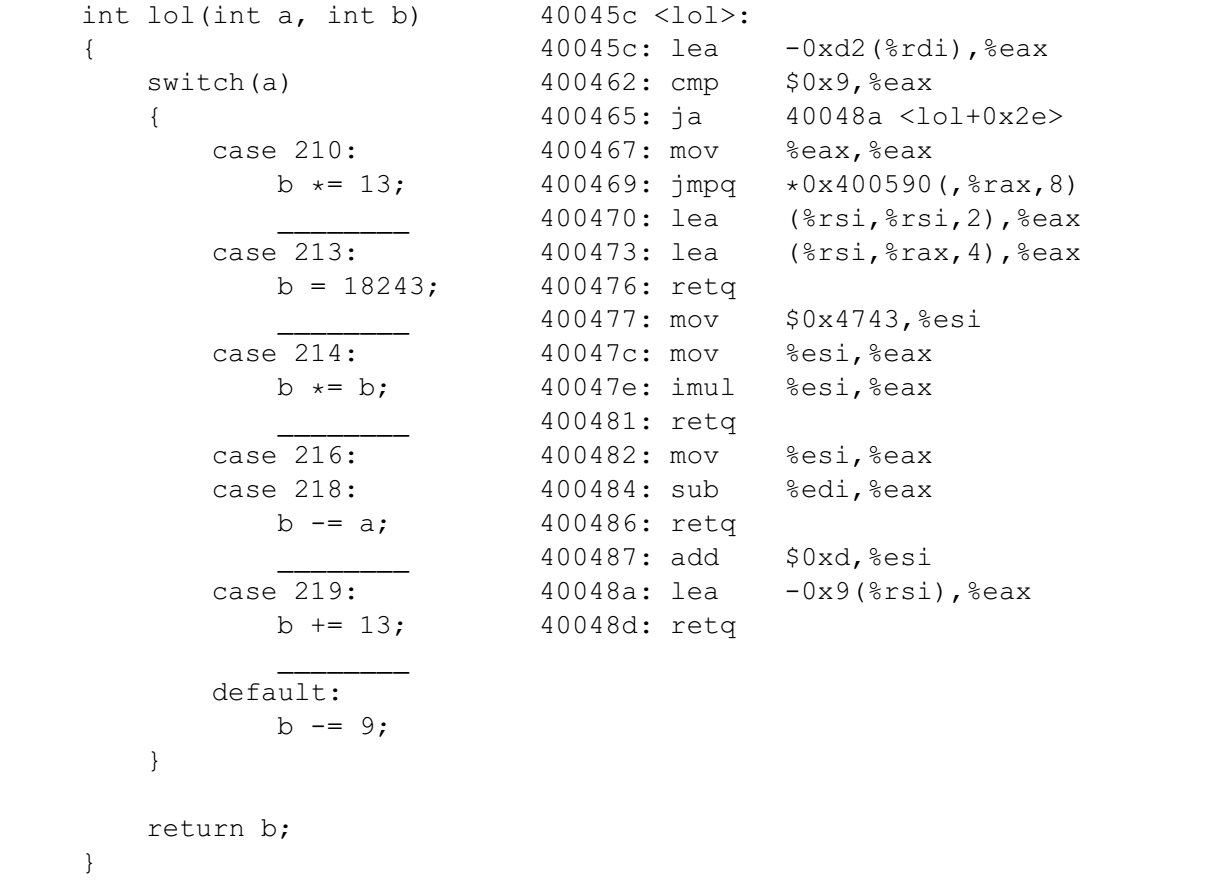
blinky: connector为起始地址+24，其指向数据的vacuum\_tube[1]偏移量为16，因此对应代码D

pinky: 返回值为其实地址+4处的值，对应代码C

clyde: mosfet地址为起始地址+8，对应代码A

### Problem 4.

Consider the following C code and assembly code:



Using the available information, fill in the jump table below. (Feel free to omit leading zeros.) Also, for each case in the switch block which should have a break, write break on the corresponding blank line.

使用可获得的信息，填写下面的跳转表。（可以省略前导零。）在c代码中添加应有的break。

Hint提示: 0xd2 = 210 and 0x4743 = 18243.

0x400590: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 0x400598: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x4005a0: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 0x4005a8: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x4005b0: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 0x4005b8: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x4005c0: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 0x4005c8: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x4005d0: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 0x4005d8: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

汇编代码：40045c：a-210；400465：不在0~9内则跳转到default即40048a；

case 中没有的211、212、215、217，对应跳转表中1、2、5、7，为default地址即40048a；

400469：按跳转表跳转；

400470~400476：返回b\*13，对应c代码中case 210（有break），即跳转表0位置处应为400470；

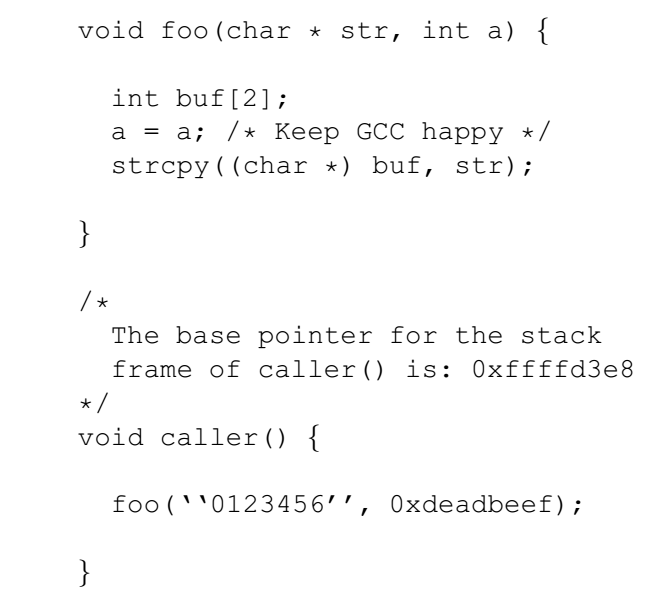
400477~400481：返回18243的平方，对应case 213（无break），40047c对应case 214（有break），即跳转表位置3处为400477，位置4处为40047c；

400482~400486：返回b-a，对应case 216、218（有break），即跳转表6、8处应为400482；

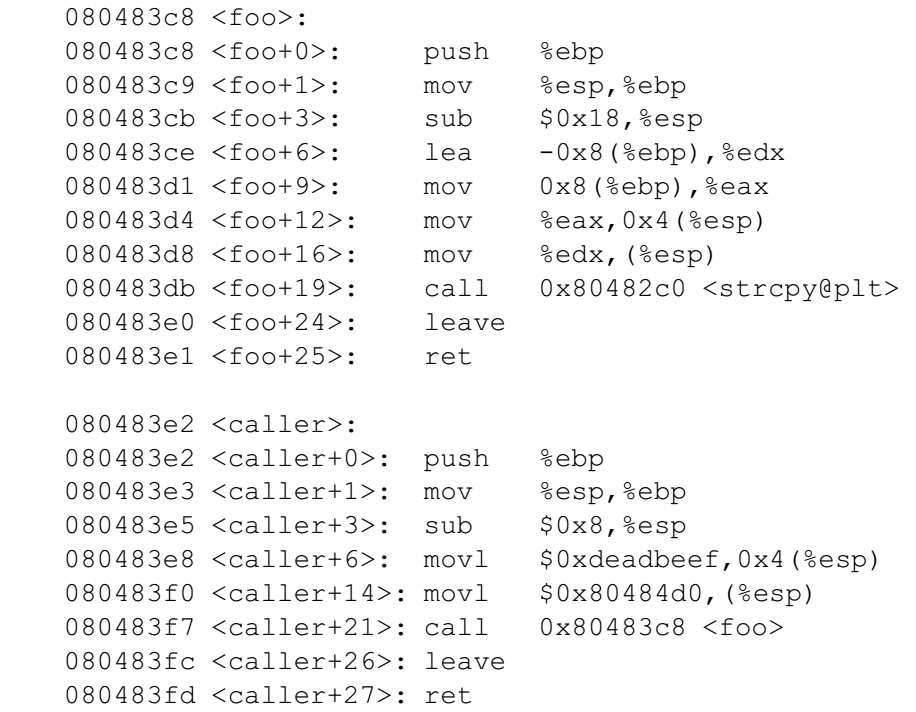
400487：计算b+13，对应case 219；然后-9返回，对应default，因此无break，跳转表9处为400487

### Problem 5.

This problem concerns the following C code, compiled on a 32-bit machine:



Here is the corresponding machine code on a 32-bit Linux/x86 machine:

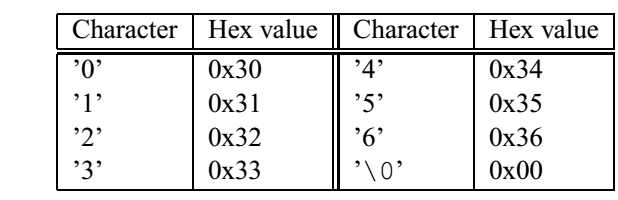


Here are some notes to help you work the problem:

• strcpy(char \*dst, char \*src) copies the string at address src (including the terminating ‘\0’ character) to address dst.

• Keep endianness in mind.

• You will need to know the hex values of the following characters:



Now consider what happens on a Linux/x86 machine when caller calls foo.

1. Just before foo calls strcpy, what integer x, if any, can you guarantee that buf[x] == a ?

foo调用strcpy之前，使得buf[x] == a的整数x值为多少？

1. At what memory address is the string “0123456” stored (before it is strcpy’d)?

在被strcpy拷贝之前，字符串“0123456”保存在哪个地址处？

1. Just after strcpy returns to foo, fill in the following with hex values:

strcpy返回到foo之后，在答题页中填写各地址处的十六进制值。

1. Immediately before the call to strcpy, what is the the value at %ebp (not what is %ebp)?

strcpy调用前，%ebp处的值是？

1. Immediately before foo’s ret call, what is the value at %esp (what’s on the top of the stack)?

foo中ret指令调用前，%esp处的值是？

1. Will a function that calls caller() segfault or notice any stack corruption? Explain.

调用caller()的函数会段错误或栈损坏吗？请解释。

依次画出caller及foo的栈帧。以四字节为单位，依次是：

caller:

old ebp; a; strAddr; foo retAddr

foo:

caller ebp; buf[1]; buf[0]; null; null; strAddr; bufAddr

A: 要使int数据类型（4字节）的buf[x] == a，沿栈帧从buf[0]开始往上找，直到a的位置，应该是buf[5]

B: 从caller的汇编代码可以看出，字符串地址为0x80484d0

C: strcpy返回到foo之后，buf的8字节空间中被写入了’0’到’\0’这8个字符的Hex Value.

buf[0] = 0x 33 32 31 30 字符串的前4个字符

buf[1] = 0x 00 36 35 34 字符串的后4个字符

buf[2] = 0x ff ff d3 e8 caller ebp的值在c代码中有提示

buf[3] = 0x 08 04 83 fc foo返回后应执行caller的leave

buf[4] = 0x 08 04 84 d0 字符串地址

D: ebp处的值为caller ebp，即0xffffd3e8

E: ret前已执行leave，栈恢复到caller的栈帧，esp处值为foo retAddr，即0x080483fc

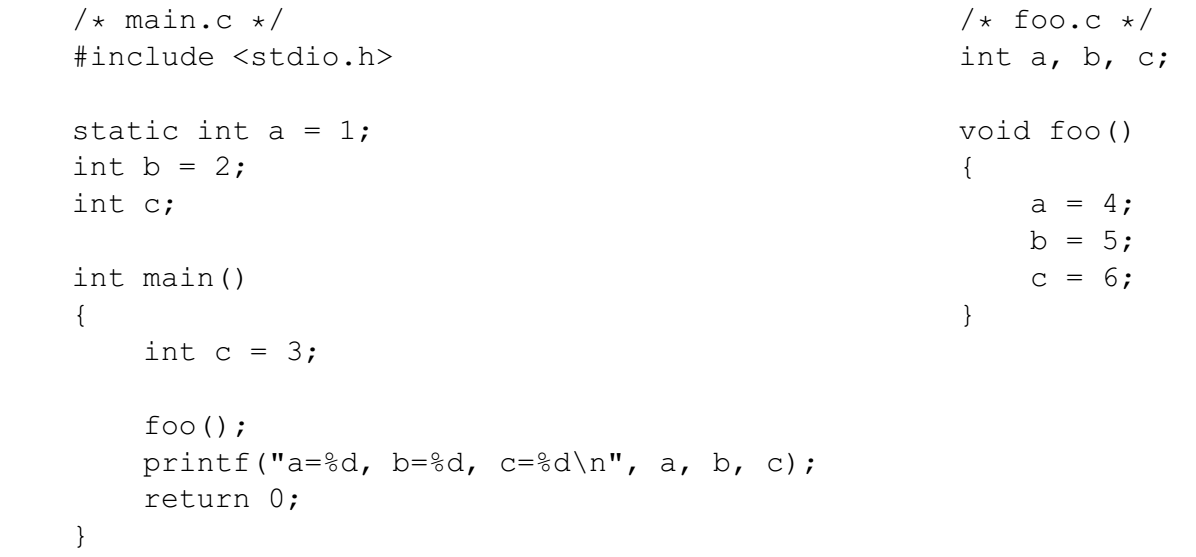
F: 不会，字符串含结束符共8字节，正好不超过buf的8字节

### Problem 6.

Consider the executable object file a.out, which is compiled and linked using the command

unix> gcc -o a.out main.c foo.c

and where the files main.c and foo.c consist of the following code:



What is the output of a.out?

a=1, b=5, c=3

a: main中，a为本地符号，不受foo中全局符号a影响；

b: 全局符号，初始值在main中定义，被foo改变；

c: 有全局符号c及main的局部变量c，printf中c为局部变量，不受foo中全局符号c影响

### Problem 7.

Make the following assumptions: 假设

• There is only one level of cache 只有一级高速缓存

• Physical addresses are 8 bits long (m = 8) 物理地址为8位

• The block size is 4 bytes (B = 4) 块大小为4字节

• The cache has 4 sets (S = 4) 高速缓存有4组

• The cache is direct mapped (E = 1) 高速缓存为直接映射

1. What is the total capacity of the cache? (in number of data bytes)

高速缓存总容量是多少字节？

1. How long is a tag? (in number of bits)

标记有几位？

1. Assuming that the cache starts clean (all lines invalid), please fill in the following tables, describing what happens with each operation.

假设高速缓存开始是空的，填表描述每个操作发生了什么。

B=4,b=2; S=4,s=2; t=8-s-b=4;

1. cache共4组，每组1行，每行4字节，因此共16字节
2. 标记位数t为4
3. 驱逐：miss且缓存非空时发生

|  |  |  |  |
| --- | --- | --- | --- |
|  | Set index? | Hit or Miss? | Eviction驱逐? |
| load 0x00 (0000 00 00) | 0 | miss | no |
| load 0x04 (0000 01 00) | 1 | miss | no |
| load 0x08 (0000 10 00) | 2 | miss | no |
| store 0x12 (0001 00 10) | 0 | miss | yes |
| load 0x16 (0001 01 10) | 1 | miss | yes |
| store 0x06 (0000 01 10) | 1 | miss | yes |
| load 0x18 (0001 10 00) | 2 | miss | yes |
| load 0x20 (0010 00 00) | 0 | miss | yes |
| store 0x1A (0001 10 10) | 2 | hit | no |

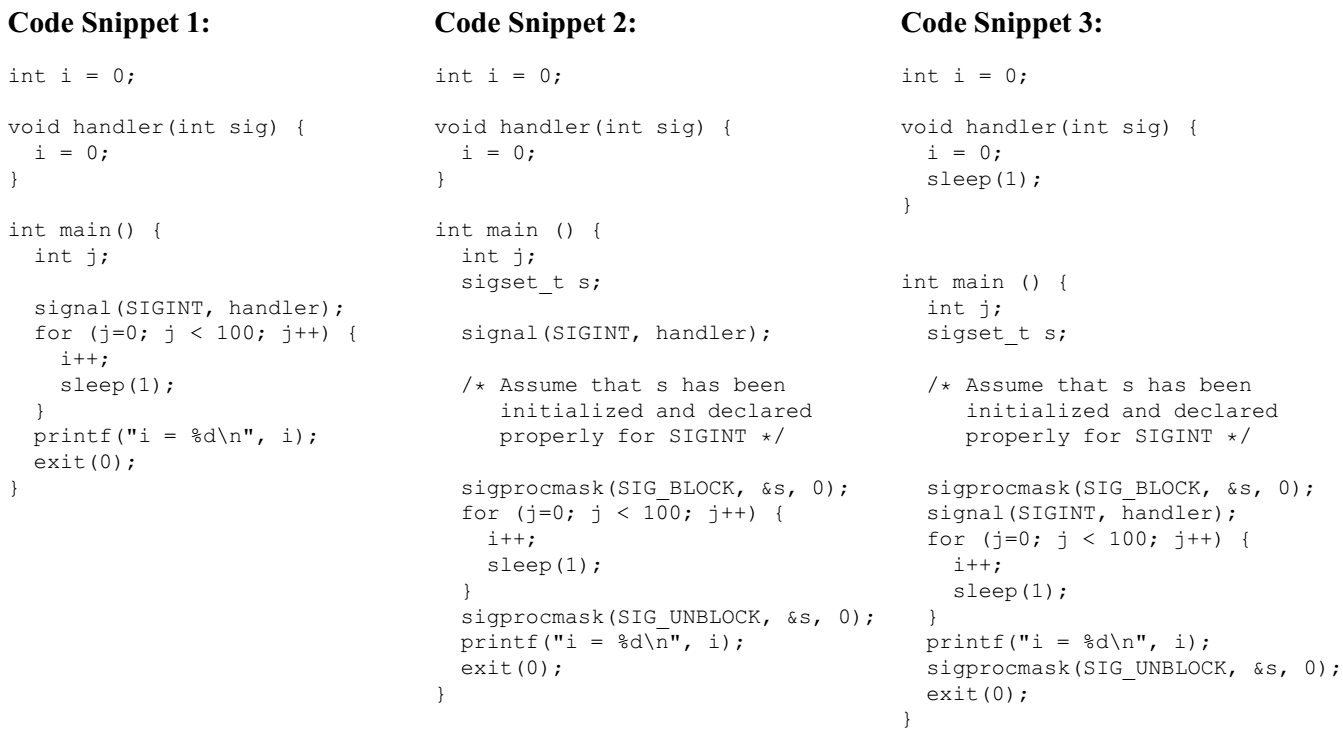
### Problem 8.

Consider the following three different snippets of C code. Assume that an arbitrary number of SIGINT signals, and only SIGINT signals, can be sent to the code snippets randomly from some external source.

考虑下面3段C代码，假设任意数量的SIGINT信号，并且只有SIGINT信号，可以从某个外部源随机发送到代码段。

What are the values of i that could possibly be printed by the printf command at the end of each program?

各段代码打印的i值可能有哪些？

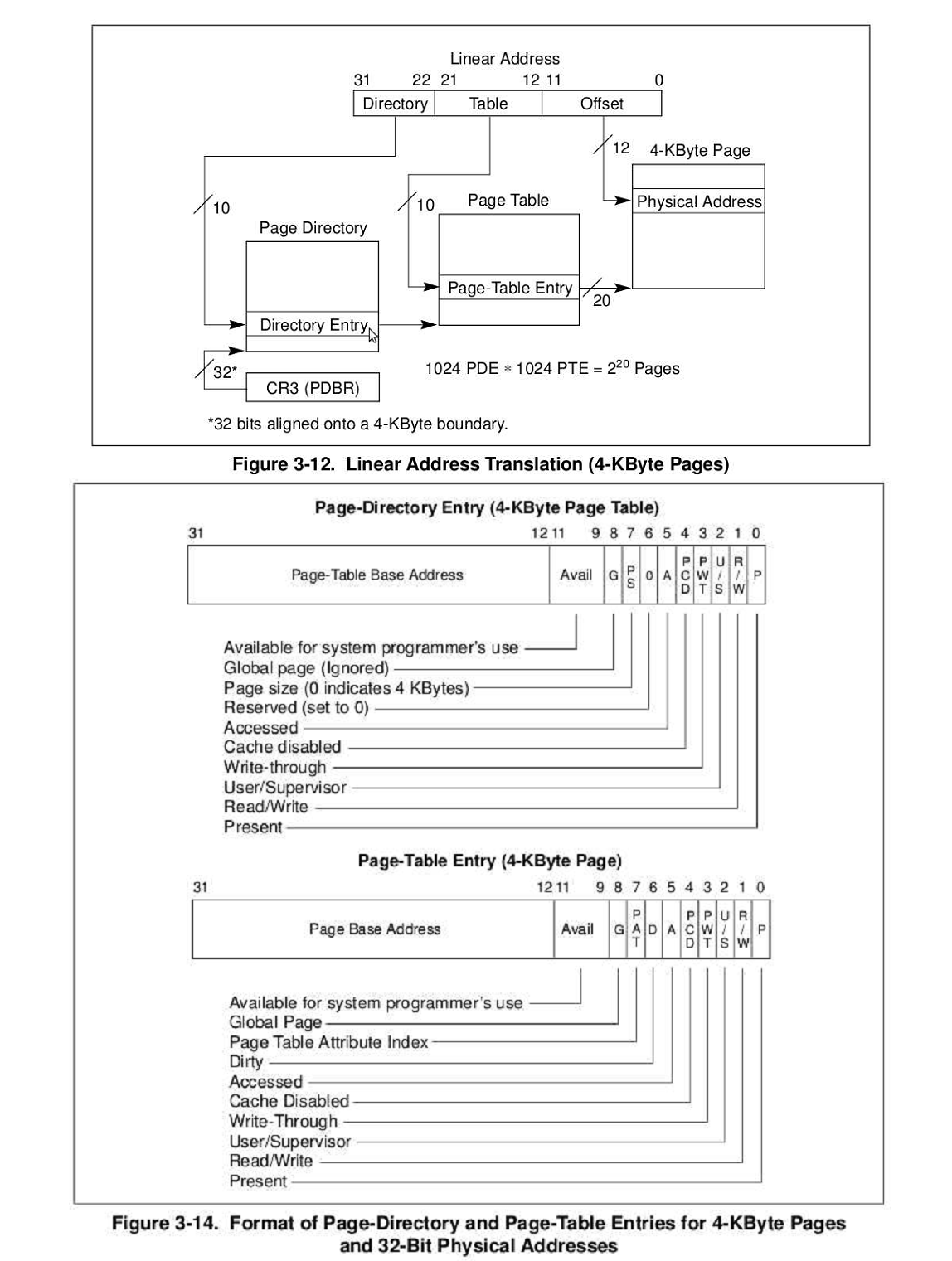


1. 主程序中，i自增100次，但随时可能会因SIGINT信号被重置为0；因此i从0~100都有可能
2. SIGINT信号被block后i开始自增，100次后i=100，然后unblock。在printf前如果有SIGINT信号来，那么i=0；如果没有信号来，那么i=100；因此i可能取值有两个：0, 100
3. 信号block之后i自增至100，然后执行printf，此时信号仍block中，因此i取值只有100.

### Problem 9.

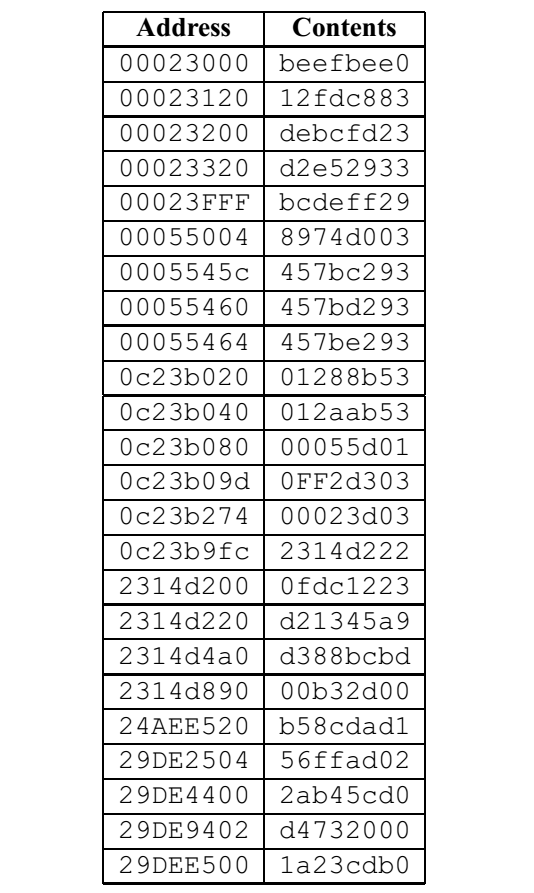
This problem deals with virtual memory address translation using a multi-level page table, in particular the 2-level page table for a 32-bit Intel system with 4 KByte pages tables.

这道题涉及多级页表的虚拟地址翻译，使用32位Intel系统，4 KB页表。



The contents of the relevant sections of memory are shown on this table. Any memory not shown can be assumed to be zero. The Page Directory Base Address is 0x0c23b000.

下表给出了相关部分的存储器内容。没有显示的存储器假设为零。页目录基地址为0x0c23b000。



For each of the following problems, perform the virtual to physical address translation. If an error occurs at any point in the address translation process that would prevent the system from performing the lookup, then indicate this by writing “FAILURE” and noting the physical address of the table entry that caused the failure.

对下面每个问题，进行虚拟地址到物理地址翻译。如果过程中发生错误，则写“FAILURE”和发生错误的表条目的物理地址。

For example, if you were to detect that the present bit in the PDE is set to zero, then you would leave the PTE address in (b) empty, and write “FAILURE” in (c), noting the physical address of the offending PDE.

例如，如果你检测到PDE有效位为零，那么(b)中的PTE为空，(c)中写“FAILURE”，记下违规的PDE条目的物理地址。

1. Read from virtual address 0x080016ba:

VPO = 0x6ba, VPN = 0x08001, VPN1 = 0x020, VPN2 = 0x001

* 1. Physical address of PDE: \_0x0c23b000 + 4 \* VPN1 = 0x0c23b080\_

ps: VPN1 = 0x080>>2; 4 \* VPN1 = VPN1<<2，相当于VPN高12位的最后2位清零。

PDE: 0x00055d01, P=1: valid, PTBA: 0x00055000

* 1. Physical address of PTE: \_0x00055000 + 4 \* VPN2 = 0x00055004\_

PTE: 0x8974d003, P=1: valid, PPN:0x8974d

c. Success: The physical address accessed is \_0x8974d6ba\_

or

Failure: Address of table entry causing failure is \_\_\_\_\_\_\_\_\_\_\_

2. Read from virtual address 0x9fd28c10:

VPO = 0xc10, VPN = 0x9fd28, VPN1 = 0x27f, VPN2 = 0x128

4 \* VPN1 = 0x9fc（VPN高12位最后2位清零）

1. Physical address of PDE: \_0x0c23b000 + 4 \* VPN1 = 0x0c23b9fc\_

PDE: 0x2314d222, P=0: invalid

b. Physical address of PTE: \_\_\_\_\_\_\_\_\_\_\_

c. Success: The physical address accessed is \_ FAILURE \_

or

Failure: Address of table entry causing failure is \_0x0c23b9fc \_