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_____

20

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

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_____ _____ _____ _____

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).

Operation	Set index?	Hit or Miss?	Eviction?
load 0x00 $(0000\ 0000)_2$	0	miss	no
load 0x04 $(0000\ 0100)_2$		miss	
load 0x08 $(0000\ 1000)_2$			
store 0x12 $(0001\ 0010)_2$	0		
load 0x16 $(0001\ 0110)_2$			
store 0x06 $(0000\ 0110)_2$			
load 0x18 $(0001\ 1000)_2$	2		
load 0x20 $(0010\ 0000)_2$			
store 0x1A $(0001\ 1010)_2$			

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 $-T_{Min}$? 补码中，-TMin 的值是多少？
(a) T_{Min}
(b) T_{Max}
(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？

(a) 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。

(b) 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。

(c) 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。

(d) 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

13. 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？

(a) 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。

(b) 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。

(c) 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。

(d) 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 $1.11 * 2^{3} = 1110 = 14$

前 3 个数在规格化数范围内：$1.00*2^{0}$ $1.10*2^{3}$ $1.011*2^{3}$

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。

```
char* inky(struct ms_pacman *ptr){
  return &(ptr->transistor.bjt);
}

long blinky(struct ms_pacman *ptr){
  return ptr->connector->
         transistor.vacuum_tube[1];
}

int pinky(struct ms_pacman *ptr){
  return ptr->resistor;
}

int clyde(struct ms_pacman *ptr){
  return ptr->transistor.mosfet;
}
```

A
```
mov  0x8(%rdi), %rax
retq
```

B
```
lea  0x8(%rdi), %rax
retq
```

C
```
mov  0x4(%rdi), %eax
retq
```

D
```
mov  0x18(%rdi),%rax
mov  0x10(%rax),%rax
retq
```

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:

```
int lol(int a, int b)
{
    switch(a)
    {
        case 210:
            b *= 13;
            ________
        case 213:
            b = 18243;
            ________
        case 214:
            b *= b;
            ________
        case 216:
        case 218:
            b -= a;
            ________
        case 219:
            b += 13;
            ________
        default:
            b -= 9;
    }

    return b;
}
```

```
40045c <lol>:
40045c: lea    -0xd2(%rdi),%eax
400462: cmp    $0x9,%eax
400465: ja     40048a <lol+0x2e>
400467: mov    %eax,%eax
400469: jmpq   *0x400590(,%rax,8)
400470: lea    (%rsi,%rsi,2),%eax
400473: lea    (%rsi,%rax,4),%eax
400476: retq
400477: mov    $0x4743,%esi
40047c: mov    %esi,%eax
40047e: imul   %esi,%eax
400481: retq
400482: mov    %esi,%eax
400484: sub    %edi,%eax
400486: retq
400487: add    $0xd,%esi
40048a: lea    -0x9(%rsi),%eax
40048d: retq
```

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:

```
void foo(char * str, int a) {

  int buf[2];
  a = a; /* Keep GCC happy */
  strcpy((char *) buf, str);

}

/*
  The base pointer for the stack
  frame of caller() is: 0xffffd3e8
*/
void caller() {

  foo(``0123456'', 0xdeadbeef);

}
```

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

```
080483c8 <foo>:
080483c8 <foo+0>:     push   %ebp
080483c9 <foo+1>:     mov    %esp,%ebp
080483cb <foo+3>:     sub    $0x18,%esp
080483ce <foo+6>:     lea    -0x8(%ebp),%edx
080483d1 <foo+9>:     mov    0x8(%ebp),%eax
080483d4 <foo+12>:    mov    %eax,0x4(%esp)
080483d8 <foo+16>:    mov    %edx,(%esp)
080483db <foo+19>:    call   0x80482c0 <strcpy@plt>
080483e0 <foo+24>:    leave
080483e1 <foo+25>:    ret

080483e2 <caller>:
080483e2 <caller+0>:  push   %ebp
080483e3 <caller+1>:  mov    %esp,%ebp
080483e5 <caller+3>:  sub    $0x8,%esp
080483e8 <caller+6>:  movl   $0xdeadbeef,0x4(%esp)
080483f0 <caller+14>: movl   $0x80484d0,(%esp)
080483f7 <caller+21>: call   0x80483c8 <foo>
080483fc <caller+26>: leave
080483fd <caller+27>: ret
```

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:

Character	Hex value	Character	Hex value
'0'	0x30	'4'	0x34
'1'	0x31	'5'	0x35
'2'	0x32	'6'	0x36
'3'	0x33	'\0'	0x00

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

A. 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？

B. 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？

C. 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。

D. 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？

E. 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？

F. 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:

```
/* main.c */
#include <stdio.h>

static int a = 1;
int b = 2;
int c;

int main()
{
    int c = 3;

    foo();
    printf("a=%d, b=%d, c=%d\n", a, b, c);
    return 0;
}
```

```
/* foo.c */
int a, b, c;

void foo()
{
    a = 4;
    b = 5;
    c = 6;
}
```

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

(a) 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？

(b) How long is a tag? (in number of bits)
标记有几位？

(c) 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;

a) cache 共 4 组，每组 1 行，每行 4 字节，因此共 16 字节

b) 标记位数 t 为 4

c) 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?

Code Snippet 1:

```
int i = 0;

void handler(int sig) {
  i = 0;
}

int main() {
  int j;

  signal(SIGINT, handler);
  for (j=0; j < 100; j++) {
    i++;
    sleep(1);
  }
  printf("i = %d\n", i);
  exit(0);
}
```

Code Snippet 2:

```
int i = 0;

void handler(int sig) {
  i = 0;
}

int main () {
  int j;
  sigset_t s;

  signal(SIGINT, handler);

  /* Assume that s has been
     initialized and declared
     properly for SIGINT */

  sigprocmask(SIG_BLOCK, &s, 0);
  for (j=0; j < 100; j++) {
    i++;
    sleep(1);
  }
  sigprocmask(SIG_UNBLOCK, &s, 0);
  printf("i = %d\n", i);
  exit(0);
}
```

Code Snippet 3:

```
int i = 0;

void handler(int sig) {
  i = 0;
  sleep(1);
}


int main () {
  int j;
  sigset_t s;

  /* Assume that s has been
     initialized and declared
     properly for SIGINT */

  sigprocmask(SIG_BLOCK, &s, 0);
  signal(SIGINT, handler);
  for (j=0; j < 100; j++) {
    i++;
    sleep(1);
  }
  printf("i = %d\n", i);
  sigprocmask(SIG_UNBLOCK, &s, 0);
  exit(0);
}
```

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。

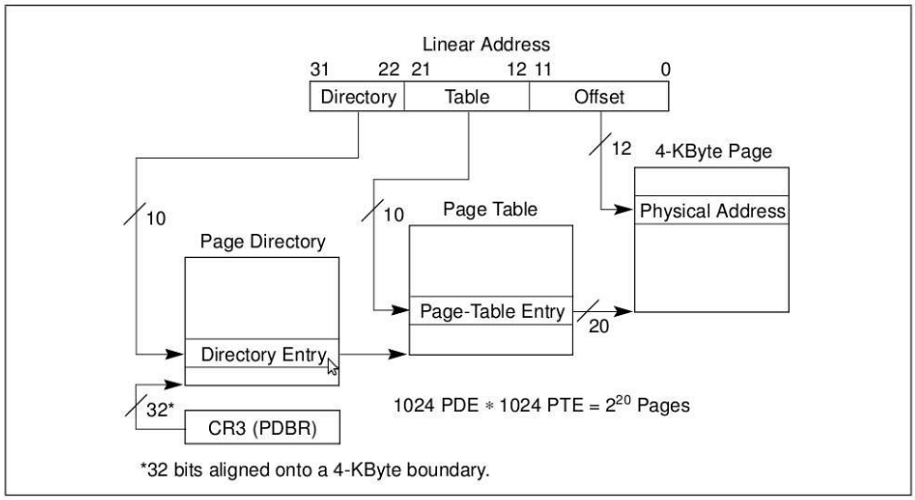

Figure 3-12. Linear Address Translation (4-KByte Pages)

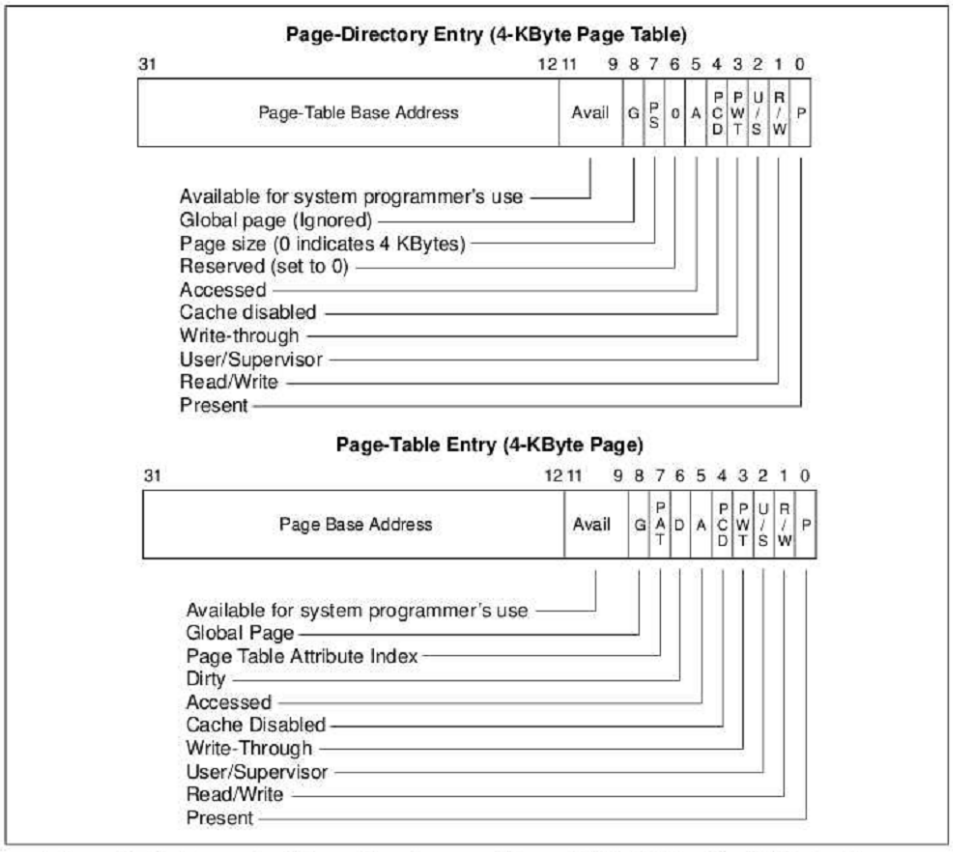

Figure 3-14. Format of Page-Directory and Page-Table Entries for 4-KByte Pages and 32-Bit Physical Addresses

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。

Address	Contents
00023000	beefbee0
00023120	12fdc883
00023200	debcfd23
00023320	d2e52933
00023FFF	bcdeff29
00055004	8974d003
0005545c	457bc293
00055460	457bd293
00055464	457be293
0c23b020	01288b53
0c23b040	012aab53
0c23b080	00055d01
0c23b09d	0FF2d303
0c23b274	00023d03
0c23b9fc	2314d222
2314d200	0fdc1223
2314d220	d21345a9
2314d4a0	d388bcbd
2314d890	00b32d00
24AEE520	b58cdad1
29DE2504	56ffad02
29DE4400	2ab45cd0
29DE9402	d4732000
29DEE500	1a23cdb0

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

a. 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

b. 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）

a. 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__