

装  
订  
线  
内

不  
要  
答  
题

# 北京大学信息科学技术学院考试试卷

考试科目： 计算机系统导论 姓名： \_\_\_\_\_ 学号： \_\_\_\_\_

考试时间： 2014 年 11 月 13 日 任课教师： \_\_\_\_\_

题号	一	二	三	四	五	六	七	八	总分
分数									
阅卷人									

## 北京大学考场纪律

- 1、考生进入考场后，按照监考老师安排隔位就座，将学生证放在桌面上。无学生证者不能参加考试；迟到超过 15 分钟不得入场。在考试开始 30 分钟后方可交卷出场。
- 2、除必要的文具和主考教师允许的工具书、参考书、计算器以外，其它所有物品（包括空白纸张、手机、或有存储、编程、查询功能的电子用品等）不得带入座位，已经带入考场的必须放在监考人员指定的位置。
- 3、考试使用的试题、答卷、草稿纸由监考人员统一发放，考试结束时收回，一律不准带出考场。若有试题印制问题请向监考教师提出，不得向其他考生询问。提前答完试卷，应举手示意请监考人员收卷后方可离开；交卷后不得在考场内逗留或在附近高声交谈。未交卷擅自离开考场，不得重新进入考场答卷。考试结束时间到，考生立即停止答卷，在座位上等待监考人员收卷清点后，方可离场。
- 4、考生要严格遵守考场规则，在规定时间内独立完成答卷。不准交头接耳，不准偷看、夹带、抄袭或者有意让他人抄袭答题内容，不准接传答案或者试卷等。凡有违纪作弊者，一经发现，当场取消其考试资格，并根据《北京大学本科考试工作与学术规范条例》及相关规定严肃处理。
- 5、考生须确认自己填写的个人信息真实、准确，并承担信息填写错误带来的一切责任与后果。
- 学校倡议所有考生以北京大学学生的荣誉与诚信答卷，共同维护北京大学的学术声誉。

以下为试题和答题纸，共 14 页。

得分

第一题 单项选择题（每小题 2 分，共 32 分）

1、假设下列 unsigned 和 int 数均为 32 位，

unsigned x = 0x00000001;

int y = 0x80000000;

int z = 0x80000001;

以下表达式正确的是

A.  $(-1) < x$

B.  $(-y) > -1$

C.  $\sim y + y == -1$

D.  $(z << 4) > (z * 16)$

答：( )

答案：C

A 错误 signed (-1) 和 unsigned x 比较，都按照 unsigned，所以强制类型转换后 (-1) 很大

B 错误 int 中 0x80000000 的相反数还是自己，是最小的负数

C 正确 相等关系； $\sim y + y == 0xFFFFFFFF$  是 -1 的补码表示

D 错误 应该是相等关系 unsigned, signed 左移 4 位 相当于 \*16

2、下面说法正确的是：

A. 数 0 的反码表示是唯一的

B. 数 0 的补码表示不是唯一的

C. 1000, 1111, 1110, 1111, 1100, 0000, 0000, 0000 表示唯一的整数是 0x8FEFC000

D. 1000, 1111, 1110, 1111, 1100, 0000, 0000, 0000 如果是单精度浮点表示，则表示的是  $-(1.110111111)_2 * 2^{31-127}$

答：( )

答案：D

考察反码、补码

A. 错误。若用反码表示，则可表示为 00000000 或 11111111。

B. 错误。数 0 的补码表示是唯一的：+0 的补码=+0 的反码=+0 的原码=00000000；0 的原码和反码有两种，补码只有一种。

C. 错误。如果是无符号数表示，则表示的是 **0x8FEFC000**；如果是补码表示，则表示的是 **-0x70104000**

D. 正确。

3、下面表达式中为“真”的是：

A. (unsigned) -1 < -2

B. 2147483647 > (int) 2147483648U

C. (0x80005942 >> 4) == 0x09005942

D. 2147483647 + 1 != 2147483648

答：( )

**答案：B**

4、下列的指令组中，那一组指令只改变条件码，而不改变寄存器的值？

A. CMP, SUB

B. TEST, AND

C. CMP, TEST

D. LEAL, CMP

答：( )

**答案：C**

**SUB 和 AND 都同时会改变条件码和寄存器的值，LEAL 不改变改变条码。**

5、下列指令中，寻址方式不正确的是

A. MOVB %ah, 0x20(, %ecx, 8)

B. LEAL (0xA, %eax), %ebx

C. SUBB 0x1B, %bl

D. INCL (%ebx, %eax)

答：( )

**答案：B**

**存储器数的基址应该存放在一个基址寄存器中。**

6、有如下定义的结构，在 x86-64 下，下述结论中错误的是？

```
struct {  
    char c;  
    union {  
        char vc;
```

- ```

        double value;
        int vi;
    } u;
    int i;
} sa;

```
- A. `sizeof(sa) == 24`  
 B. `(&sa.i - &sa.u.vi) == 8`  
 C. `(&sa.u.vc - &sa.c) == 8`  
 D. 优化成员变量的顺序，可以做到“`sizeof(sa) == 16`”

答: ( )

**答案: B**

由于对齐的需求，在 **x86-64** 下，要保证 **sa.u.value** (**double** 类型变量) 的地址必须 8 字节对齐，而在 **ia32** 的 **Linux** 系统中，可 4 字节对齐。故 **sizeof(sa)** 在 **x86-64** 下要占用三个 8 字节的空间共 24 字节。**sa.u.vc** 与 **sa.c** 之间和 **sa.i** 与 **sa.u.vi** 之间的地址值之差均为 8，但即使不知道 **(&sa.i - &sa.u.vi)** 表示之间可以放置多少个整数 (值为 2)，也可通过优化成员变量顺序后有 **sizeof(sa) == 16**，用排除法得出正确答案。

7、关于如何避免缓冲区溢出带来的程序风险，下述错误的做法为？

- A. 编程时定义大的缓冲区数组  
 B. 编程时避免使用 `gets`，而采用 `fgets`  
 C. 程序运行时随机化栈的偏移地址  
 D. 在硬件级别引入不可执行代码段的机制

答: ( )

**答案: A**

**B、C、D** 均为讲义中所提及的解决缓冲区溢出风险的方法。**A** 策略则无法从根本上解决缓冲区溢出问题，只要输入足够长数据就仍然可以实现缓冲区溢出攻击。

8、对简单的 `switch` 语句常采用跳转表的方式实现，在 **x86-64** 系统中，下述最有可能正确的 `switch` 分支跳转汇编指令为哪个？

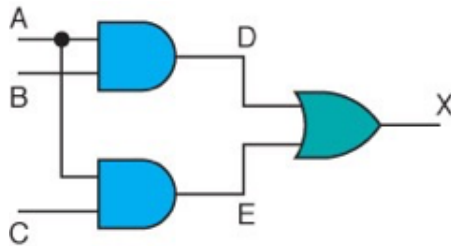
- A. `jmp .L3(, %eax, 4)`  
 B. `jmp .L3(, %eax, 8)`  
 C. `jmp *.L3(, %eax, 4)`  
 D. `jmp *.L3(, %eax, 8)`

答：( )

答案：D

A 与 B 都是错误的基于跳转表跳转的指令正确格式。C、D 的指令格式正确，但在 x86-64 系统中，每个地址占 8 个字节，因此更有可能的答案是 D。

9、对应下述组合电路的正确 HCL 表达式为



A.  $\text{Bool } X = (A \mid\mid B) \ \&\& \ (A \mid\mid C)$

B.  $\text{Bool } X = A \mid\mid (B \ \&\& \ C)$

C.  $\text{Bool } X = A \ \&\& \ (B \mid\mid C)$

D.  $\text{Bool } X = A \mid\mid B \mid\mid C$

答：( )

答案：C

10、若处理器实现了三级流水线，每一级流水线实际需要的运行时间分别为 2ns、2ns 和 1ns，则此处理器不停顿地执行完毕 10 条指令需要的时间为：

A. 21ns

B. 22ns

C. 23ns

D. 24ns

答：( )

答案：D

$2+2+2*10 = 24$

11、关于 RISC 和 CISC 的描述，正确的是：

A. CISC 指令系统的指令编码可以很短，例如最短的指令可能只有一个字节，因此 CISC 的取指部件设计会比 RISC 更为简单。

- B. CISC 指令系统中的指令数目较多, 因此程序代码通常会比较长; 而 RISC 指令系统中通常指令数目较少, 因此程序代码通常会比较短。
- C. CISC 指令系统支持的寻址方式较多, RISC 指令系统支持的寻址方式较少, 因此用 CISC 在程序中实现访存的功能更容易。
- D. CISC 机器中的寄存器数目较少, 函数参数必须通过栈来进行传递; RISC 机器中的寄存器数目较多, 只需要通过寄存器来传递参数。

答: ( )

**答案: C**

**考查对 CISC 和 RISC 基本特点的描述, A 和 B 都是描述反了, D 则是太绝对, RISC 也有可能用栈来传递参数。**

12、关于流水线技术的描述, 正确的是:

- A. 指令间数据相关引发的数据冒险, 一定可以通过暂停流水线来解决。
- B. 流水线技术不仅能够提高执行指令的吞吐率, 还能减少单条指令的执行时间。
- C. 增加流水线的级数, 一定能获得性能上的提升。
- D. 流水级划分应尽量均衡, 不均衡的流水线会增加控制冒险。

答: ( )

**答案: A**

**说明: B 会增加单条指令的执行时间, C 可能会降低性能, D 和控制冒险没有关系**

13、下面关于程序性能的说法中, 哪种是正确的?

- A. 处理器内部只要有多个功能部件空闲, 就能实现指令并行, 从而提高程序性能。
- B. 同一个任务采用时间复杂度为  $O(\log N)$  算法一定比采用复杂度为  $O(N)$  算法的执行时间短
- C. 转移预测总是能带来好处, 不会产生额外代价, 对提高程序性能有帮助。
- D. 增大循环展开 (loop unrolling) 的级数, 有可能降低程序的性能 (即增加执行时间)

答: ( )

**答案: D**

14、仅考虑以下代码, 哪些程序优化总是被编译器自动进行? (假设 `int i, int j, int A[N], int B[N], int m, int *p` 都是局部变量, `N` 是一个整数型

常量, int foo(int) 是一个函数)

| 优化前                                                        | 优化后                                                                            |
|------------------------------------------------------------|--------------------------------------------------------------------------------|
| A.   for (j = 0 ; j < N ; j<br>++)<br>B[i] *= A[j];        | int temp = B[i];<br>for (j= 0 ; j < N ; j ++)<br>temp *= A[j];<br>B[i] = temp; |
| B.   for (j = 0 ; j < N ; j<br>++)<br>m += i*N*j;          | int temp = i*N;<br>for (j= 0 ; j < N ; j ++)<br>m += temp * j;                 |
| C.   i = foo(N);<br>j = foo(N);<br>if (*p != 0)<br>m = j ; | j = foo(N);<br>if (*p != 0)<br>m = j ;                                         |
| D.   for (j = 0 ; j < foo(N) ;<br>j ++)<br>m ++;           | int temp = foo(N);<br>for (j= 0 ; j < temp ;<br>j ++)<br>m ++;                 |

答: (        )

答案: B

15、以下关于存储结构的讨论, 那个是正确的

- A. 增加额外一级存储, 数据存取的延时一定不会下降
- B. 增加存储的容量, 数据存取的延时一定不会下降
- C. 增加额外一级存储, 数据存取的延时一定不会增加
- D. 以上选项都不正确

答: (        )

答案: D

16、关于 cache 的 miss rate, 下面那种说法是错误的。

- A. 保持 E 和 B 不变, 增大 S, miss rate 一定不会增加
- B. 保持总容量和 B 不变, 提高 E, miss rate 一定不会增加
- C. 保持总容量和 E 不变, 提高 B, miss rate 一定不会增加
- D. 如果不采用“LRU”, 使用“随机替换策略”, miss rate 可能会降低

答: (        )

答案: C

注：实际答案 B 的说法也是有错误的。例如：保持总容量和 B 不变，提高 E，就意味着以前的两组可能并成一组。假设 A 和 B 组并成了新的一组。现在假设有一个访问序列  $b_1, b_2, a_1, a_2, a_3, \dots, a_n, b_1, b_2$ 。其中  $b_i$  是会被放到 B 组的块， $a_i$  是会被放到 A 组的块。n 足够大使得可以把合并后的组中的所有块都换出去。那么按合并后的情况全部都不命中，而按合并前的情况最后两次访问  $b_1, b_2$  仍然能命中。

因此，选 C 或 B 都算正确。



|    |
|----|
| 得分 |
|    |

## 第二题 (10 分)

1) 假设下列 unsigned 和 int 数均为 5 位 (有符号整型用补码运算表示), 在下表中填入正确答案 (每空 1 分, 共 6 分)

```
int y = - 7;
unsigned z = y;
```

|       | Decimal Representation | Binary Representation |
|-------|------------------------|-----------------------|
| z     |                        |                       |
| y - z |                        |                       |
| TMin  |                        |                       |

答案

|       | Decimal Representation | Binary Representation |
|-------|------------------------|-----------------------|
| z     | 25                     | 1 1001                |
| y - z | 0                      | 0 0000                |
| TMin  | -16                    | 1 0000                |

2) 请按 IEEE 浮点标准的单精度浮点数表示下表中的数值, 首先写出形如  $(-1)^s \times M \times 2^E$  的表达式, 然后给出十六进制的表示。(每格 1 分, 共 4 分)

注: 单精度浮点数的字段划分如下:

符号位 (s): 1-bit; 阶码字段 (exp): 8-bit; 小数字段 (frac): 23-bit; 偏置值 (bias): 127。

| Value | $(-1)^s \times M \times 2^E$ , $1 \leq M < 2$ | Hex representation |
|-------|-----------------------------------------------|--------------------|
| 0.375 |                                               |                    |
| -12.5 |                                               |                    |

答案

| Value | $(-1)^s \times M \times 2^E$<br>$1 \leq M < 2$ | Hex representation |
|-------|------------------------------------------------|--------------------|
| 0.375 | $1.1 \times 2^{-2}$                            | 0x3EC00000         |
| -12.5 | $(-1) * 1.1001 * 2^3$                          | 0xC1480000         |

|    |
|----|
| 得分 |
|    |

### 第三题（10 分）

阅读以下代码。假设代码运行在 IA32 的计算机上，字长为 4，请给出各个变量在内存中的十六进制字节表示（地址从小到大）。

注意 tiny\_float 是一种 8 位的浮点数，1 个符号位，4 个指数位，3 个尾数位。

```
int main()
{
    int a = 0x15213;
    unsigned char b = ((char)-5);
    tiny_float c = 19;
    float d;
    if (b < 4)
        d = -0.96875;
    else
        d = 769;
}
```

| a  |    |    |    |
|----|----|----|----|
| 13 | 52 | 01 | 00 |

| b  |
|----|
| FB |

| c  |
|----|
| 5A |

| d  |    |    |    |
|----|----|----|----|
| 00 | 40 | 40 | 44 |

每空 1 分

|    |
|----|
| 得分 |
|    |

#### 第四题（10 分）

一个函数如下，其中部分代码被隐去，请通过gdb调试信息补全代码（4分）。

```
int f(int n, int m) {
    if (m > 0) {
        if (n > 1) {
            int r = f(n - 1, m);
            return (r - 1 + m) % n + 1;
        }
        else if (n == 1) {
            return 1;
        }
    }
    return 0;
}
```

{考点：x86-64函数调用、参数传递及栈的使用。函数调用通过rdi和rsi传递第一和第二个参数，栈中只记录函数的返回地址。由于是递归调用，需要用栈保存递归过程中参数变量的值。同时考察xor、test、lea、idiv、sete等指令的使用。难点1：两个判断“n > 1”和“n == 1”在汇编代码中只有一次比较，第二次判断相等是通过sete使得n==1时返回值为1，否则返回值为0实现的。难点2：表达式“(r - 1 + m) % n + 1”比较复杂，需要综合多条语句的信息才能分析出来，并且变量r对应于f(n-1,m)的返回值即寄存器%eax，因而不存在相应的赋值指令。}

如下是通过“gcc -g -O2”命令编译后，在gdb中通过“disas f”命令得到的反汇编代码，其中有两个汇编指令不全，请补全这两条汇编指令（2分）。

```
0x00000000004004e0 <f+0>:      mov     %rbx,-0x10(%rsp)
0x00000000004004e5 <f+5>:      mov     %rbp,-0x8(%rsp)
0x00000000004004ea <f+10>:     xor     %eax,%eax
0x00000000004004ec <f+12>:     sub     $0x10,%rsp
0x00000000004004f0 <f+16>:     test    %esi,%esi
0x00000000004004f2 <f+18>:     mov     %edi,%ebp
0x00000000004004f4 <f+20>:     mov     %esi,%ebx
0x00000000004004f6 <f+22>:     jle     0x400513 <f+51>
0x00000000004004f8 <f+24>:     cmp     $0x1,%edi
```

```

0x000000000004004fb <f+27>:    jle     0x400521 <f+65>
0x000000000004004fd <f+29>:    lea     -0x1(%rbp),%edi
0x00000000000400500 <f+32>:    callq  0x4004e0 <f>
0x00000000000400505 <f+37>:    lea     -0x1(%rax,%rbx,1),%edx
0x00000000000400509 <f+41>:    mov     %edx,%eax
0x0000000000040050b <f+43>:    sar     $0x1f,%edx
0x0000000000040050e <f+46>:    idiv    %ebp
0x00000000000400510 <f+48>:    lea     0x1(%rdx),%eax
0x00000000000400513 <f+51>:    mov     (%rsp),%rbx
0x00000000000400517 <f+55>:    mov     0x8(%rsp),%rbp
0x0000000000040051c <f+60>:    add     $0x10,%rsp
0x00000000000400520 <f+64>:    retq
0x00000000000400521 <f+65>:    sete    %al
0x00000000000400524 <f+68>:    movzbl  %al,%eax
0x00000000000400527 <f+71>:    jmp     0x400513 <f+51>

```

{考察点：函数中使用到了%rbp和%rbx寄存器，两者都是callee保存的寄存器，使用前需要压栈，函数返回时需要弹栈恢复寄存器的值。通过前后汇编代码的对比，应该可以猜出两个空分别填写什么；但要注意，压栈和弹栈时，%rsp寄存器的值不同，因而对应的地址表示也不同。}

已知在调用函数  $f(4, 3)$  时，我们在函数  $f$  中指令 `retq` 处设置了断点，下面列出的是程序在第一次运行到断点处暂停时时，相关通用寄存器的值。请根据你对函数及其汇编代码的理解，填写当前栈中的内容。如果某些内存位置处内容不确定，请填写  $x$ 。（4分）

|                  |                            |
|------------------|----------------------------|
| <code>rax</code> | <code>0x1</code>           |
| <code>rbx</code> | <code>0x3</code>           |
| <code>rcx</code> | <code>0x3</code>           |
| <code>rdx</code> | <code>0x309c552970</code>  |
| <code>rsi</code> | <code>0x3</code>           |
| <code>rdi</code> | <code>0x1</code>           |
| <code>rbp</code> | <code>0x2</code>           |
| <code>rsp</code> | <code>0x7fffffff340</code> |
| <code>rip</code> | <code>0x400520</code>      |

{ 考察点：递归调用的返回地址共三处是明确的，并且相同，值可以从反汇编代码中确定（1分）；三次递归调用程序栈中，压入的 `%rbx(m)` 的值不变，压入的 `%rbp(n)` 的值为每次减小 1（1分）；注意  $x86-64$ ，栈中的数据都是 64 位的，但因为数值均比较小，所以这 9 个位置处的高 4 字节均为 0（1分）；其余位置的内容均是不确定的（1分）。 }

|                            |                         |
|----------------------------|-------------------------|
| <code>0x7fffffff38c</code> | $x$                     |
| <code>0x7fffffff388</code> | $x$                     |
| <code>0x7fffffff384</code> | $x$                     |
| <code>0x7fffffff380</code> | $x$                     |
| <code>0x7fffffff37c</code> | $x$                     |
| <code>0x7fffffff378</code> | $x$                     |
| <code>0x7fffffff374</code> | <code>0x0</code>        |
| <code>0x7fffffff370</code> | <code>0x00400505</code> |
| <code>0x7fffffff36c</code> | <code>0x0</code>        |
| <code>0x7fffffff368</code> | <code>0x4</code>        |
| <code>0x7fffffff364</code> | <code>0x0</code>        |
| <code>0x7fffffff360</code> | <code>0x3</code>        |
| <code>0x7fffffff35c</code> | <code>0x0</code>        |
| <code>0x7fffffff358</code> | <code>0x00400505</code> |
| <code>0x7fffffff354</code> | <code>0x0</code>        |
| <code>0x7fffffff350</code> | <code>0x3</code>        |
| <code>0x7fffffff34c</code> | <code>0x0</code>        |
| <code>0x7fffffff348</code> | <code>0x3</code>        |
| <code>0x7fffffff344</code> | <code>0x0</code>        |
| <code>0x7fffffff340</code> | <code>0x00400505</code> |
| <code>0x7fffffff33c</code> | <code>0x0</code>        |
| <code>0x7fffffff338</code> | <code>0x2</code>        |
| <code>0x7fffffff334</code> | <code>0x0</code>        |
| <code>0x7fffffff330</code> | <code>0x3</code>        |
| <code>0x7fffffff32c</code> | $x$                     |
| <code>0x7fffffff328</code> | $x$                     |
| <code>0x7fffffff324</code> | $x$                     |
| <code>0x7fffffff320</code> | $x$                     |

|    |
|----|
| 得分 |
|    |

### 第五题（8 分）

阅读下面的汇编代码，根据汇编代码填写 C 代码中缺失的部分，然后描述该程序的功能。

```

pushl %ebp
movl %esp,%ebp
movl $0x0, %ecx
cmpl $0x0, 8(%ebp)
jle .L1
.L2
movl $0x0, %edx
movl 8(%ebp), %eax
divl $0x0a
addl %edx, %ecx
movl %eax, 8(%ebp)
cmpl $0x0, 8(%ebp)
jg .L2
.L1
movl 0x0, %edx
movl %ecx, %eax
divl 0x3
cmpl 0x0, %edx
jne .L3
movl 0x1, %eax
jmp .L4
.L3
movl 0x0, %eax
.L4

```

```

int fun(unsigned x) {
    int bit_sum = 0;
    while ( (int) x > 0 ) {
        bit_sum += x % 10;
        x = x / 10;
    }
    if ( bit_sum % 3 == 0 )
        return 1;
    else
        return 0;
}

```

红色下划线部分为答案，每空 1 分。

该程序用来判断一个不大于  $2^n-1$  的非负整数是否为 3 的倍数，如果大于  $2^n-1$ ，则直接返回 1（3 分）。

|    |
|----|
| 得分 |
|    |

第六题（10 分）

请分析Y86 ISA中新加入的一条指令：**caddXX**，条件加法。其功能可以参考**add**和**cmovXX**两条指令。

|        |   |    |    |    |
|--------|---|----|----|----|
| caddXX | C | fn | rA | rB |
|--------|---|----|----|----|

若在教材所描述的SEQ处理器上执行这条指令，请按下表填写每个阶段进行的操作。需说明的信号包括：**icode**, **ifun**, **rA**, **rB**, **valA**, **valB**, **valC**, **valE**, **valP**, **Cnd**; the register file **R[]**, data memory **M[]**, Program counter **PC**, condition codes **CC**。其中对存储器的引用必须标明字节数。如果在某一阶段没有任何操作，请填写**none**指明。

| Stage      | caddXX rA, rB                                                                                                  |
|------------|----------------------------------------------------------------------------------------------------------------|
| Fetch      | <b>icode:ifun</b> $\leftarrow M_1[PC]$<br><b>rA:rB</b> $\leftarrow M_1[PC+1]$<br><b>valP</b> $\leftarrow PC+2$ |
| Decode     | <b>valA</b> $\leftarrow R[rA]$<br><b>valB</b> $\leftarrow R[rB]$                                               |
| Execute    | <b>valE</b> $\leftarrow valA+valB$<br><b>Cnd</b> $\leftarrow Cond(CC,ifun)$                                    |
| Memory     | <b>none</b>                                                                                                    |
| Write back | <b>if(Cnd) R[rB]</b> $\leftarrow valE$                                                                         |
| PC update  | <b>PC</b> $\leftarrow valP$                                                                                    |

（每个操作 1 分）

注：**Execute** 阶段可以写上 **if(Cnd) Set CC**，不计分，但如果只写了 **Set CC** 是要扣分的。**Write back** 阶段的操作不可以写成 **R[rB]  $\leftarrow$  Cnd? valE: valB**

|    |
|----|
| 得分 |
|    |

### 第七题 (10 分)

如下是使用 C 语言描述的链表结构的声明，链表的结尾使用空指针来表示。同时使用函数 `int length (List *p)` 来计算链表的长度。为简化起见，假设该链表是非循环的。

```
typedef struct LIST {
    struct LIST *next;
    int data;
} List;
```

- 1) 函数 `count_pos1` 用来计算链表中 `data` 为正数的元素个数，并将结果存放在地址 `k`。以下的程序可能存在问题导致效率很低或程序出错，请指出并修改。

(4 分)

```
void count_pos1 (List *p, int *k) {
    int i;
    for (i = 0; i < length(p); i++) {
        if (p->data > 0)
            *k++;
        p = p->next;
    }
}
```

- 2) 为提高程序性能，可以考虑删除变量 `i` 以消除函数调用。请修改上述程序达到该目的。(2 分)

- 3) 上述程序内层循环的汇编片段如下所示。假设该链表不为空且大部分数据都为正数，转移预测全部正确，设计中有足够多的部件来实现指令并行。其中访存操作全部 `cache` 命中，时延为 3 `cycle`，其他指令时延为 1`cycle`。请计算以下程序的 CPE 下限，并给出文字说明。(4 分)

```
.L1:
    movl    4(%eax), %ecx
    testl   %ecx, %ecx
    jle     .L2
    incl    %edx
.L2:
```



```

movl    (%eax), %eax
testl   %eax, %eax
jne     .L1

```

答案:

1)

```

void count_pos1 (List *p, int *k) {
    int l, num=0, len;
    len = length(p)
    for (i = 0; i < len; i++) {
        if ( p->data  > 0)
            num++;
        p = p->next;
    }
    *k = num
}

```

( 4 分)

2) while(p) 或其他相同功能的语句 (2 分)

3) L1 和 L2 的代码没有数据依赖，完全可以并行。(2 分)

CPE 的下限为 3+1=4。(2 分)

|    |
|----|
| 得分 |
|    |

### 第八题（10 分）

假设存在一个能够存储四个 Block 的 Cache，每一个 Block 的长度为 2Byte。假设内存空间大小共是 16Byte，即内存空间地址长度一共是 4bit，可访问地址为(0~15)，数据访问地址序列如下所示，访问数据单位是 Byte，默认替换策略是 LRU。

2 3 10 9 6 8

1) 如果 Cache 的结构是下图所示（S=2， E=2），请在下图空白处填入访问上述六次数据访问后 Cache 的状态。注：用[0-1]表示地址 0 至 1 上对应的数据（4 分）

|      | V | TAG | Block | V | TAG | Block   |
|------|---|-----|-------|---|-----|---------|
| set0 | 1 | 10  | [8-9] | 0 |     |         |
| set1 | 1 | 01  | [6-7] | 1 | 10  | [10-11] |

2) 这六次数据访问一共产生了多少次 Miss \_\_\_\_\_（2 分）

答案：4

3) 如果 Cache 的替换策略改成 MRU（即，最近使用的数据被替换出去），请在下图空白处填入访问上述六次数据访问后 Cache 的状态（2 分）。

|      | V | TAG | Block | V | TAG | Block |
|------|---|-----|-------|---|-----|-------|
| set0 | 1 | 10  | [8-9] | 0 |     |       |
| set1 | 1 | 00  | [2-3] | 1 | 01  | [6-7] |

4) 这六次数据访问一共产生了多少次 Miss \_\_\_\_\_（2 分）

答案：4