1、编译优化的策略有哪些？请说明各种优化策略能提高程序运行速度的原理。（至少5种不同策略）

要点：（1）不仅要说策略是什么，而且要说出背后的原理；

（2）出题的本意是想问问，利用哪些硬件特性来提速，即优化后能更好地发挥硬件的作用，加快运行速度。计算机系统是计算机软件和硬件组成的整体。单纯地从软件层面介绍优化（如去掉没有用的废代码减少了执行指令的数目；算出可以计算的表达式的值而不产生相应的机器指令等等），这就与硬件特性无关了。

参考答案：

（1）**循环展开：将程序执行流程变成一个顺序结构。消除引起循环的跳转指令，使指令流水线利用更充分，避免在指令流水线上产生要被丢弃的“半成品”而浪费时间。**

（2）**有分支语句向无分支语句转换：使用条件传送 cmov* 等指令，可以提高指令流水线的利用率，原因同(1) [可参见 “L03_Intel 中央处理器.pdf：流水线的控制分支冒险”]。**

（3）**调整指令执行顺序：后面的指令用到前面指令的结果，前面的指令结果还未产生，后面的指令就要等待，产生阻塞就会影响指令流水线的速度。调整指令顺序的目的是减少可能的阻塞 [可参见 “L03_Intel 中央处理器.pdf：流水线的数据冒险”]。**

（4）**使用执行速度更快的机器指令：例如，将一个变量中的内容乘 2，可以用变量自己与**

自己相加，也可以用左移运算。不同指令的执行速度不同，使用速度更快的指令代替完成相同功能的慢速指令，会提高速度。

注意：此处指的是一条指令被另一条指令所代替；还不是用多条机器指令来代替一条慢速指令的意思，这种一对多的优化，有一点算法优化的味道。

SIMD 串流操作 并行优化

(5) 使用串操作指令代替用循环一个数据一个数据的处理（传送、比较、串置初值等等）：串操作指令产生的根源就是加快速度。

(6) 使用SIMD：一条指令成组操作，节约了操作次数。

(7) 使用位数更长的寄存器：使用字节数更大的寄存器，一次就可以处理更多的内容，充分利用硬件中已有数据线宽度；

(8) 对一个二维数组调整数据处理顺序（如按列序操作调整为按行序操作）：提高CPU中cache 的命中率，减少cache与内存之间来回的数据交换，从而节约时间。

(9) 变量与寄存器绑定：访问变量变成访问绑定的对应寄存器，访问寄存器的速度要快于访问内存（包括cache）的速度，（无需地址计算，虚地址向物理地址转换等等操作，因而要快）。

(10) 并行优化：利用多线程、多核等特性。

2、为了提高程序的执行速度，在编写C语言程序时，可进行哪些优化（不考虑编译器的优化）？（至少给出5种优化场景，可举例说明）

要点：(1) 不考虑编译器的优化，意思在编译器优化开关关闭时，生成的执行程序运行速度要

快。

（2）优化算法是会提高速度，但我们这门课主要是介绍基本的C语句的执行过程，算法方面不应考虑；此处的优化是指写程序时应该注意的问题。

（3）是使用 C 写程序，而不是使用机器指令编写程序，像一条C语句对应的一段机器指令中可能产生的优化不在考虑之列。

参考答案：

（1） 优化数据的访问顺序，如在for循环里对于二维数组，按照先行序，再列序访问每个元素。

（2）减少重复计算，比如for(int i=0;i<strlen(a);i++)中strlen(a)多次计算。

（3）调用封装了串操作指令的函数，如memcpy，memset、memcmp等。

（4）变递归程序为迭代程序，函数调用传递参数，断点压栈等多种操作，既慢又有栈溢出的风险。

（5）用移位实现乘除法运算，比如x*2变为x<<1。

（6）调整条件语句中组合条件的子条件顺序。例如 if （A && B），假设 90%的情况下A 会成立，10%的情况下 B成立，就应该写成 if (B && A) , 在 90%的情况下，减少了对条件 A 的判断。

(7) 封装了SIMD 指令的函数调用

(8) 多线程的利用

(9) 当然，写程序时，可以做一些编译器可以优化的工作，如去掉废代码；

（10）有一些优化是编译器无法无做到的（也可以说是目前的编译器还没有特别聪明），比如，与指针相关的数据访问。

3、阅读下面的程序，回答问题。

```
.section  .data
```

```
    array:  .long 10, -20, 30, -40, 50
    length = (. -array)/4        # length 为array中
元数的个数，= 5
    format:  .ascii "%d\n\0"
.section  .text
.global  _start
_start:
    mov  $0,  %eax
    mov  $length, %ecx
    lea  array,  %edi    # ①
lp_1:
    cmpl  $0,  (%edi)
    jl    lp_2          # ②
    inc   %eax
lp_2:
    add  $4,  %edi
    sub  $1,  %ecx        # ③
    jne lp_1
    push  %eax
    push  $format
    call   printf
    mov  $1,  %eax    # 程序正常退出
    mov  $0,  %ebx
    int  $0x80
```

1） 上述程序的功能是什么？运行后，屏幕上显示的是什么？

统计array数组中非负数的个数并显示。 显示 3

2） 若标号 lp_1 写到 ①处语句前，程序运行的结果是什么？为什么？

显示 5。每次循环都将array的地址送 edi, 每次循环都是判断数组的第一个元素是否为负数。

3） 若将 ② 处的语句改为 “jb lp_2”,程序运行的结

果是什么?

显示 5。jb 是无符号数比较转移，任何无符号数都不低于 0。

4） 若漏写了 ③ 处的语句 ，程序运行会出现什么现象？为什么?

程序运行异常终止。表面上，%edi 在循环中不断加4，加到值为0时，循环终止。但是随着edi的增加，cmpl $0, (%edi),访问的内存单元超出程序空间范围，引起异常。

要点提示:

（1） cmpl $0, (%edi) 中，源操作数是 0，目的操作数是 (%edi) , 执行的是减法，被减数为 目的操作数，根据 (%edi) – 0 设置标志位。可以简单理解被 目的操作数与源操作数的比较，看比较结果如何。

（2） 要说出原因，而不能只写一个 显示的结果。