**第8次作业**

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

**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：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。**

**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和算法层面的优化）？ （至少给出 5种优化场景，可举例说明）**

**3.** 分析优化题。下面的 C语言程序段（32位段）实现了一个数组求和的功能。给出了编译后的调试版本的汇编语言代码（斜体部分为C语句）。

***int sum = 0;***

00FB24C8 mov dword ptr [ebp-30h], 0

***for (i = 0; i < 5; i++)***

00FB24CF mov dword ptr [ebp-8], 0

00FB24D6 jmp 00FB24E1

00FB24D8 mov eax, dword ptr [ebp-8]

00FB24DB add eax, 1

00FB24DE mov dword ptr [ebp-8], eax

00FB24E1 cmp dword ptr [ebp-8], 5

00FB24E5 jge 00FB24F6

***sum += a[i];***

00FB24E7 mov eax, dword ptr [ebp-8]

00FB24EA mov ecx, dword ptr [ebp-30h]

00FB24ED add ecx, dword ptr [ebp+eax\*4-24h]

00FB24F1 mov dword ptr [ebp-30h], ecx

00FB24F4 jmp 00FB24D8

00FB24F6 … …

(1) 指出该段程序执行效率不高的原因。

(2) “00FB24E5 jge 00FB24F6”处指令的机器码为7DH 0FH，解释0FH代表的含义。

(3) 保留循环结构，改编相应的汇编语言程序，以提高程序的执行效率。要求写出变量与寄存器对应关系（可以用标号来代表指令地址）。

(4) 用循环展开的方法（即去除循环）优化程序段。

(5) 用一条语句实现：将（eax）\* 5 + 10的结果送到ebx，不用考虑溢出。

**4.** 程序分析题。阅读下面的程序，回答问题。

.section .data

array: .long 10, -20, 30, -40, 50

length = (. -array) / 4 # length = 5 为 array 中元数的个数

format: .ascii "%d\n\0"

.section .text

.global \_start

\_start: mov $0, %eax

mov $length, %ecx

lea array, %edi # ①

L1: cmpl $0, (%edi)

jl L2 # ②

inc %eax

L2: add $4, %edi

sub $1, %ecx # ③

jne L1

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？

(2) 若标号L1写到 ① 处语句前，程序运行的结果是什么？为什么？

(3) 若将 ② 处的语句改为 “jb L2”,程序运行的结果是什么？

(4) 若漏写了 ③ 处的语句，程序运行会出现什么现象？为什么？

**5.** 程序分析题。阅读下面的程序（部分语句列出了反汇编时看到的结果），回答问题。

.686P

.model flat, c

exitProcess proto stdcall :dword

includelib kernel32.lib

includelib libcmt.lib

includelib legacy\_stdio\_definitions.lib

printf proto :vararg

.data

lpfmt db "%s --> %d", 0dh, 0ah, 0

value dd 0

string db "123", 0

.stack 200

.code

main proc c

push offset string ; 00108280 push 179010h

call atoui ; 00108285 call 001082B1

add esp, 4 ; 0010828A add esp,4

mov value, eax ; 0010828D mov dword ptr ds:[0017900Ch], eax

invoke printf, offset lpfmt, offset string, value

invoke ExitProcess, 0

main endp

atoui proc

push ebp ; 001082B1 push ebp

mov ebp, esp

push edx

push esi

mov esi, [ebp+8] ; **①**

mov eax, 0 ; **②**

atoi\_convert:

mov dl, [esi]

cmp dl, 0

jz atoi\_convert\_over

sub dl, 30h

movzx edx, dl

imul eax, 10

add eax, edx

inc esi

jmp atoi\_convert

atoi\_convert\_over:

pop esi

pop edx

pop ebp

ret

atoui endp

end

1. 上述程序运行后，屏幕上显示的结果是什么？
2. 子程序atoui的功能是什么？它的入口参数和出口参数分别是什么？
3. 若标号atoi\_convert 写在②处语句之前，即有 atoi\_convert: mov eax, 0，程序的运行后，屏幕上显示的结果是什么？

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. 画出执行到 ① 处语句时堆栈的示意图，要求在单元外标明单元的地址，并画出ESP指向的单元。

设开始执行程序时，寄存器窗口看到如下内容：

EAX = 0017AF34 EBX = 00307000

ECX = 00000000 EDX = C10E52E8

ESI = 00636BD0 EBP = 004FF9D8

ESP = 004FF994 EIP = 00108280

**6.** 程序改错。在一个以0结束的字符串中，将所有的大写字母转换为对应的小写字母，并将转换结果输出。请将程序中的语法错误和逻辑错误圈出来，并在其右侧写出正确的形式（重点关注带\*的行）。

.686P

.model flat, c

ExitProcess proto stdcall :dword

printf proto :ptr sbyte, :vararg

includelib kernel32.lib

includelib libcmt.lib

includelib legacy\_stdio\_definitions.lib

.data

fmt db "%s" , 0

buf db "Assembly Language" ; \*

.code

main proc

mov edi, buf ; \*

loop\_start: mov dl, dword ptr [edi] ; \*

cmp dl, ‘0’ ; \*

jz exit

cmp dl, 'A'

jbe loop\_end ; \*

cmp [edi], 'Z' ; \*

jg loop\_start ; \*

add dl, 'a'-'A'

mov buf, dl ; \*

loop\_end: inc edi

jne loop\_start ; \*

exit: invoke printf, offset fmt, buf ; \*

invoke ExitProcess, 0

main endp

end

**7.** 程序填空。下面程序的功能是：找出buf中最大和最小的数，分别保存到max\_v、min\_v中。

... ...

.data

buf dd 10, 30, -25, -10, 23, -20, 20

n = \_\_\_\_\_\_\_\_\_\_\_ ; n为buf 中有符号双字类型数据的个数

max\_v dd 0

min\_v dd 0

.code

main proc c

mov eax, \_\_\_\_\_\_\_ ; buf 中的第0个数，作为当前的最大数

mov edx, eax

mov \_\_\_\_ , n - 1

mov esi, \_\_\_\_\_\_\_\_

L1: mov ebx, \_\_\_\_\_\_\_\_

cmp eax, ebx

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mov eax, ebx

L2: cmp edx, ebx

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mov edx, ebx

L3: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dec ecx

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mov max\_v, eax

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

invoke ExitProcess, 0

main endp

end