**第5次作业**

**1.** 编写汇编语言程序段（Intel）去优化下面的C函数，使得汇编程序中没有分支转移语句。假设变量p的地址为 (ebp) –10h。

int getv(int \*p)

{

return p ? \*p : 0;

}

**2.** 编写汇编语言程序段（Intel）实现C语言的！运算：(eax) = !(ebx)，要求没有分支转移语句。

**3.** 假设 (ax) = 0D000h, (bx) = 2000h，执行

cmp ax, bx

ja L1

程序可以转到L1吗？将 ja 换成 jg、js，结果如何？

**4.** 在分支转移指令的机器码中，如何表示转移的目标地址？

**5.** 编写与“loope lp”等效的程序段（Intel）。

**6.** 根据下列程序段（Intel）

… …

MOV ESI, OFFSET BUF1

MOV EDI, OFFSET BUF2

MOV ECX, n

LOOLA: MOV AL, [ESI]

MOV [EDI], AL

INC ESI

INC EDI

LOOP LOOPA ;使用ECX计数

… …

绘制流程图，并回答如下(1)至(5)问题。其中BUF1、BUF2均为串长度为n 的字节存储区首址。

(1) 该程序段完成了什么工作？

(2) 若将 “MOV ECX, n”误写成“MOV ECX, 0”，则循环体被执行多少次？

(3) 若漏掉了 “MOV ECX, n”，则循环体执行的次数能确定吗？为什么？

(4) 若漏掉了指令“INC ESI”，则程序运行结果如何？

(5) 若不小心将标号LOOPA上移了一行，即将标号写在指令“MOV ECX, n”之前，则程序运行情况如何？

**7.** 编写子程序PROG（不能用带变量说明的高级伪指令去定义PROG），实现如下功能：(X - Y) / Z（X、Y、Z是有符号数，不需考虑X-Y的溢出），子程序将商保存到AX、余数保存到DL。主程序（Intel）的调用格式如下：

.model flat, stdcall

… …

X DW ? ;X为2个字节的有符号整数

Y DB ? ;Y为1个字节的有符号整数

Z DB ? ;Z为1个字节的有符号整数

… …

MOV AL, Y

MOV AH, Z

PUSH AX

PUSH X

CALL PROG

**ADD ESP, 4**

… …

**8.** 阅读下面的（C和Intel汇编）程序，回答问题

void func ( )

{ unsigned short us = 65535;

unsigned int ui;

short s = -1;

int i;

int x = 0, y = 0; ; ①

ui = us;

i = s; ; ②

if (ui > 0) x = 1;

if (i > 0) y = 1; ; ③

ui = ui >> 1; ; >>1 右移一个二进制位

i = i >> 1; ; ④

ui = ~ui; ; 按位取反

i = -i; ; ⑤

}

**⑴ 执行完** ① 处语句后，

us = 0x \_\_\_ s = 0x \_\_\_\_（2个字节的16进制数）

**⑵ 执行完** ② 处语句后，

ui = 0x \_\_ i = 0x \_\_（4个字节的16进制数）

ui = us; 对应的机器指令： \_\_ eax, us、 mov ui, eax

i = s; 对应的机器指令： \_\_ eax, s、 mov i, eax

**⑶ 执行完 ③** 处语句后，

x = \_\_\_\_ y = \_\_\_\_

if (ui>0) x=1; 对应的机器指令： cmp ui, 0、 \_\_\_\_ lp1、 mov x, 1、 lp1: …

if (i >0) y=1; 对应的机器指令： cmp i, 0、 \_\_\_\_ lp2、 mov y, 1、 lp2: …

**⑷ 执行完 ④** 处语句后，

ui = 0x \_\_\_ i = 0x \_\_ （4个字节的16进制数）

ui = ui >> 1; 对应的机器指令： \_\_\_\_ ui, 1

i = i >> 1; 对应的机器指令： \_\_\_\_ i, 1

**⑸ 执行完** **⑤** 处语句后，

ui = 0x \_\_ i = 0x \_\_\_\_ （4个字节的16进制数）

ui = ~ui; 对应的机器指令： \_\_\_\_ ui

i = -i; 对应的机器指令： \_\_\_\_ i

**9.** 在VS2019（或其他VS版本）、x86平台、Debug模式，观察下面C程序的反汇编程序、内存等，分析switch … case的执行过程，并回答如下问题：

(1) 写出break对应的汇编指令；

(2) 若少写break，导致的后果是什么？

(3) 各个分支摆放顺序对结果有无影响？default 分支能不能写在最开头？

(4) 程序是如何判断并进行相应分支转移的？

#include <stdio.h>

int main( )

{ int x = 3;

int y = -1;

int z;

char c;

c = getchar(); //从键盘输入一个字符

switch (c) {

case '9':

case '7':

z = x + y;

break;

case '1':

case '0':

z = x - y;

break;

default:

z = 0;

}

printf("%d %c %d = %d \n", x, c, y, z);

return 0;

}

**10.** 在Linux环境下，对一个C语言程序进行编译、链接、调试运行，程序片段如下。

int fadd(int a, int b)

{ int temp;

temp = a + b;

return temp;

}

void main( )

{ int x = 0x1234;

int y = -32;

int result = 0;

char msg[6] = "abc12";

result = fadd(x, y);

result = \*(int \*)(msg + 1);

}

调试时，设变量x的地址（&x）为0xffffd508，y的地址（&y）为0xffffd50c，result的地址为0xffffd510，数组msg的起始地址为 0xffffb516。

(1) 执行到result = fadd(x, y) 时，写出以字节为写出内存的内容（用16进制数的形式填空，最左边是内存地址）。

0xffffd508 \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_

0xffffd510 \_\_ \_\_ \_\_ \_\_ XX XX \_\_ \_\_

0xffffd518 \_\_ \_\_ \_\_ \_\_ XX XX XX XX

(2) 数据传送指令解读

int x = 0x1234对应的机器指令为：movl $0x1234, -0x20(%ebp)，执行该语句时， ebp = 0x \_\_\_\_。

int result = 0对应的反汇编指令为 \_\_\_\_。

执行 “result = \*(int \*)(msg+1);”后， result中的值为 0x\_\_\_\_ 。

(3) 函数调用语句解读

语句result = fadd(x, y) 对应的反汇编代码（最左边的是机器指令的地址）如下。

0x56556210 <+72>: push -0xlc(%ebp)

0x56556213 <+75>: push -0x20(%ebp)

0x56556216 <+78>: call 0x5655619d <fadd>

0x5655621b <+83>: add $0x8, %esp

0x565562le <+86>: mov %eax, -0x18(%ebp)

设执行result = fadd(x, y) 之前，esp的值为0xffffd500。

① 在表格的适当位置填写刚进入函数fadd内部时，堆栈中存放的相关数据；

② 刚进入函数fadd内部时，esp = 0x\_\_\_\_；

③ 执行 “ add $0x8, %esp ” 之后，esp = 0x\_\_\_\_。

|  |  |
| --- | --- |
|  | 0xffffd4ec |
|  | 0xffffd4f0 |
| 0x5655621b | 0xffffd4f4 |
| 0x00001234 | 0xffffd4f8 |
| 0xFFFFFFE0 | 0xffffd4fc |
| XXXXXXXX | 0xffffd500 |

(4) fadd函数的反汇编代码：

0x5655619d <+0>: push %ebp

0x5655619e <+1>: mov %esp, %ebp

0x565561a0 <+3>: sub $0x10, %esp

0x565561a3 <+6>: mov 0x8(%ebp), %edx

0x565561a6 <+9>: mov 0xc(%ebp), %eax

0x565561a9 <+12>: add %edx, %eax

0x565561ab <+14>: mov %eax, -0x4(%ebp)

0x565561ae <+17>: mov -0x4(%ebp), %eax

0x565561b1 <+20>: ……………….

① 函数参数 a的地址（即&a）是 0x\_\_\_\_。

② 局部变量 temp的地址（即&temp）是 0x\_\_\_\_。

③ 执行 “add %edx, %eax” 后， CF=\_\_\_\_, SF=\_\_\_\_, ZF=\_\_\_\_, OF= \_\_\_\_。

④ 在函数的结束处，有程序段

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

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

ret

执行后可返回到调用函数的断点处。

**11.** 编写完整汇编程序（Intel）：主程序调用C函数scanf(“%s”)从键盘接受一个无符号10进制数字符串，然后调用子程序str10\_to\_num（需要编写）将这个10进制字符串转换为一个长整数，接着调用子程序num\_to\_str16（需要编写）将这个长整数转换为16进制字符串，最后主程序调用C函数printf(“%s”)在显示器上输出这个16进制字符串。例如：



要求：(1) 不能使用子程序定义和调用的伪指令（如 f1 proc v1:dword，invoke 100）；

(2) 调用子程序时，只能采用堆栈传递参数。