******华中科技大学计算机科学与技术学院2023~2024第一学期**

解答内容不得超过装订线

**“ 计算机系统基础 ”考试试卷(B卷)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **考试方式** | **闭卷** | **考试日期** | **2024-3-18** | **考试时长** | **150 分钟** |
| **专业班级** |  | **学 号** |  | **姓 名** |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **题号** | **一** | **二** | **三** | **四** | **五** | **总分** | **核对人** |
| **分值** | 20 | 20 | 20 | 20 | 20 | 100 |  |
| **得分** |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| **分 数** |  |
| **评卷人** |  |

1. **数据的存储表示和访问（共20分）**
2. 数据的存储表示（10分）

void func1()

{

int x = 10;

int y[2] = { 20,32 };

char msg[6] = "abcde";

int\* p = &x;

short z = \*(short\*)(msg + 1);

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

}

设 x的地址(&x) 为：0xffffd508； y[0]的地址为：0xffffd50c； msg[0] 的地址为：0xffffd51a；

p的地址(&p) 为：0xffffd514； z 的地址为：0xffffd518； u的地址为：0xffffd520。

在函数func1执行结束返回前，以字节形式观察内存内容。以16进制的形式填写程序用到的内存单元内容（无关的字节中的内容不要填写，打XX）。

0xffffd508 \_0A\_\_ \_00\_\_ \_00\_\_ \_00\_\_ \_14 \_ \_00\_\_ \_00\_\_ \_00\_\_

0xffffd510 \_20\_\_ \_00\_ \_ 00\_\_ \_ 00\_\_ \_08\_\_ \_D5\_\_ \_FF\_\_ \_FF\_\_

0xffffd518 \_62\_\_ \_63\_\_ \_\_61\_ \_62\_\_\_ \_63\_\_ \_64\_\_ \_65\_\_ \_00\_\_

0xffffd520 \_62\_\_ \_63\_\_ \_64\_\_ \_65\_\_ \_XX\_ \_XX\_ \_XX\_ \_XX\_\_

2、调试一个C程序时，在函数的反汇编中看到源程序语句与对应的反汇编语句如下。根据观察到的信息填空。（10分）

*int a[2][5];*

*int i = 1;*

movl $1, -0x2c(%ebp)

i 的地址为 0x0019feb8， ebp = 0x \_\_0019FEE4\_\_\_\_\_\_

*int j = 3;*

movl $3, -0x30(%ebp) &j = 0x\_\_0019FEB4\_\_\_\_\_\_\_\_

*a[i][j] = 10;*

imul $0x14, -0x2c(%ebp), %eax

执行后，eax = \_\_0x14\_\_\_\_\_\_\_\_

eax中值的含义是\_数组a中，前i-1行元素的字节长度\_\_\_

lea -0x28(%ebp, %eax, 1), %ecx

ecx中存放的值的含义是\_数组元素a[i][0]的地址\_\_

mov -0x30(%ebp), %edx

movl $0xa, (%ecx, %edx, 4)

指令中的 edx 乘4的原因是\_数组元素类型为int，字节长度为4\_

a[0][0] 的地址表达形式是 \_\_-0x28(%ebp)\_(或 -0x14(%ecx))\_;

*global[i] = 18;*  global 为一个全局整型数组

mov -0x2c(%ebp), %eax

movl $0x12, 0x417120(, %eax, 4)

global 数组的起始地址是\_0x417120\_\_\_\_,

*int\* p = &global[1];*

mov $4, %eax

shl $0, %eax

add $0x417120, %eax

mov %eax, -0x34(%ebp)

变量p 中的值是 0x\_417124\_\_­­­\_\_\_\_\_\_

*\*p = 19;*

mov -0x34(%ebp), %eax

movl $0x13, (%eax)

第二条汇编语句访存的寻址方式是\_寄存器间接寻址\_\_\_\_\_

|  |  |
| --- | --- |
| **分 数** |  |
| **评卷人** |  |

1. **程序运行的基本过程（共20分）**

1、在对某程序进行调试时，看到如下一段信息（10分，每空1分）：

① 设当前 eip 为 0x00411650，该处指令为“83 7D FC 00 cmpl $0, -4(%ebp)”，在取出该指令并进行译码后，eip = \_0x00411654\_\_\_\_\_；

② 在0x00411654处，有指令 “7E 08 jle 0041165E”，综合前一条指令，就是在\_-4(%ebp)小于等于0\_条件下，将0x0041165E →eip ,计算出该地址值的方法是 直接寻址 ；

③ 在0x0041165C 处，有jmp指令“EB 08”，该指令对应的反汇编语句是：\_\_jmp 0x00411666\_\_\_；执行该指令时，会\_\_\_无\_\_（有、无）条件跳转到 0x00411666处；

④ 在0x00411666处，有一子程序直接调用指令“E8 65 FC FF FF callq 004112D0”。由此推断子程序的入口地址是 0x4112D0 ；指令中 FFFFFC65的含义是是 子程序入口地址和当前地址的相对位移 ；执行该call指令时，CPU会将 0x 0041166B 压入堆栈中；

⑤ 在0x00411672处，有指令“FF 55 F4 callq -0xc(%ebp) ”，得到子程序入口地址的方法是 在一个局部变量中保存有子程序入口地址 ；

⑥ 执行RET指令时，CPU会 将栈顶元素 →eip。

2、函数调用（10分）

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 fmax(int a, int b) {

004115A5 push %ebp

004115A6 mov %esp,%ebp

004115A8 sub $0x10,%esp

***int result;***

***if (a > b)***

004115A9 mov 0x8(%ebx), %eax

004115AC cmp 0xc(%ebp), %eax

004115AF jle fmax+19h (04115B9h)

***result = a;***

004115B1 mov 0x8(%ebp), %eax

004115B4 mov %eax, -0x4(%ebp)

004115B7 jmp fmax+1Fh (04115BFh)

***else result = b;***

004115B9 mov 0xc(%ebp), %eax

004115BC mov %eax, -0x4(%ebp)

***return result;***

004115BF mov -0x4(%ebp), %eax

}

004115C2 mov %ebp,%esp

004115C4 pop %ebp

004115C5 ret

void main( ) { ………

int x = fmax(10, -10);

00413863 push $0xfffffff6h

00413865 push $0xa

00413867 call 004115A5

0041386C add $8, %esp

0041386F mov %eax, -0x4(%ebp)

………

}

设执行main函数中的 call 004115A5 之前，esp= 0xffffd500; ebp= 0xffffd510.

|  |  |
| --- | --- |
|  | 0xffffd4f0 |
| 0x0041386c | 0xffffd4f4 |
| 0xa | 0xffffd4f8 |
| 0xfffffff6 | 0xffffd4fc |
| XXXXXXXX | 0xffffd500 |

填写刚进入函数fmax 时（执行push %ebp 之前），堆栈中相关单元的内容。

函数参数 a的地址（即&a）是 0x\_ffffd4f8\_\_\_。

局部变量 result的地址是 0x\_ffffd4ec\_\_\_\_。

|  |  |
| --- | --- |
| **分 数** |  |
| **评卷人** |  |

1. **C语句的转换 （共20分）**

**1、阅读下面的程序，回答问题 （10分，每小题2分）。**

void func ()

{ unsigned short us\_x = 0x8000;

unsigned int ui\_x;

short s\_x = -0x8000;

int i\_x;

int u = 0, v = 0; // ①

ui\_x = us\_x;

i\_x = s\_x; // ②

if (ui\_x > 0) u = -1;

if (i\_x > 0) v = -1; // ③

ui\_x = ui\_x >> 1;

i\_x = i\_x >> 1; // ④

ui\_x = ~ui\_x;

i\_x = -i\_x; // ⑤

}

**⑴ 执行完** ① 处语句后，

us\_x = 0x \_\_8000\_\_\_\_ s\_x = 0x \_8000\_\_\_\_（2个字节的16进制数）

**⑵ 执行完** ② 处语句后，

ui\_x = 0x \_00008000\_\_\_\_ i\_x = 0x \_ffff80000\_\_\_\_\_\_\_\_（4个字节的16进制数）

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

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

**⑶ 执行完 ③** 处语句后，

u = \_\_-1\_\_\_\_\_\_\_\_\_\_ v = \_\_0\_\_\_\_\_\_\_\_\_\_\_

if (ui\_x >0) u=-1; 对应的机器指令： cmp $0, ui\_x、 \_\_jbe(jna)\_\_ process2、

mov $-1, u、 process2:…

if (i\_x >0) v=-1; 对应的机器指令： cmp $0, i\_x、 \_\_\_jle(jng)\_ process3、

mov $-1, v、 process3:…

**⑷ 执行完 ④** 处语句后，

ui\_x = 0x \_00004000\_\_\_\_\_ i\_x = 0x \_FFFFC000\_\_\_\_\_\_\_\_\_\_\_\_\_ （4个字节的16进制数）

ui\_x = ui\_x >> 1; 对应的机器指令： \_\_shrl\_ $1, ui\_x

i\_x = i\_x >> 1; 对应的机器指令： \_\_sarl\_ $1, i\_x

**⑸**  **执行完**  ⑤ 处语句后，

ui\_x = 0x \_\_FFFFBFFF\_\_\_\_\_\_\_\_\_\_\_\_ i\_x = 0x \_00004000\_\_\_\_\_ （4个字节的16进制数）

ui\_x = ~ui\_x; 对应的机器指令： \_\_notl\_\_ ui\_x

i\_x = -i\_x; 对应的机器指令： \_\_negl\_\_ i\_x

1. **程序优化 （10分）。**

设有如下一段C语言程序，请写出完成相同功能、执行速度尽可能快的汇编语言程序代码片段，并指出优化中利用了CPU中的哪些特性。

int a[5],i,sum=0;

……

for (i = 0;i < 5;i++) sum += a[i];

#eax：存放sum的值

#ecx：存放i的值

mov $0, %eax

add a(0,%ecx, 4)

mov $0, %ecx

label\_a:

add a(, %ecx, 4), %eax

add a+4(, %ecx, 4), %eax

add a+8(, %ecx, 4), %eax

add a+12(, %ecx, 4), %eax

add a+16(, %ecx, 4), %eax

mov %eax, sum

将变量和寄存器绑定，利用寄存器代替内存变量，提高速度

利用CPU中的指令流水线，将循环展开，消除循环，提高速度

|  |  |
| --- | --- |
| **分 数** |  |
| **评卷人** |  |

**四、程序阅读与理解（20分）**

**1、阅读下面的程序，回答问题 （10分）。**

**.section .data**

**array: .long -1, -2, 11, -9, 5, 100**

**length = (. -array)/4 # length 为array中元数的个数，= 6**

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

**.section .text**

**.global \_start**

**\_start:**

**mov $0, %eax**

**mov $length, %ecx**

**mov $0, %esi # ①**

**lp\_1:**

**cmpl $10, array( , %esi, 4)**

**jg lp\_2 # ②**

**inc %eax**

**lp\_2:**

**inc %esi**

**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？（2分）

统计并显示数组array中，不大于10的元素个数。

运行后，屏幕上显示4。

1. 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？（3分）

程序运行结果为6，因为始终在比较array的第一个元素-1，比较了6次，每次都不大于10。

1. 若将 **②** 处的语句改为 “ja lp\_2”,程序运行的结果是什么？(2分)

程序运行结果为1。

1. 若漏写了 ③ 处的语句 ，程序运行会出现或什么现象？为什么？（3分）

则程序进入死循环，最终异常终止。因为%ecx的值不变，程序会一直循环。但%esi的值会不断增大，最终cmpl $10, array( , %esi, 4) 访问内存超出范围，会引起异常退出。

**2. 程序填空（10分，每空 1分）**

将一个以0结束的字符串中所有的大写字母转换为对应的小写字母，并将转换后的字符串、以及修改的字母个数输出。

……

.section .data

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

buffer: .ascii "Computer System Foundation \n\_\0\_\_\_"

.section .text

.global \_start

\_start:

mov \_$0\_\_\_\_\_\_\_\_, %ebx # ebx 存放buffer 数组元素的下标

mov \_$0\_\_\_\_\_\_\_\_, %ecx # ecx 存放修改字母的个数

loop\_begin:

mov buffer(%ebx), %al

cmp $0, %al

\_je loop\_over\_\_\_\_\_\_\_\_\_\_\_\_\_

cmp $‘A’, %al

\_jb next\_char\_\_\_\_\_\_\_\_\_\_\_\_\_

cmp $’Z’, %al

\_ja next\_char\_\_\_\_\_\_\_\_\_\_\_\_\_

add \_$’a’ - $‘A’\_\_\_\_\_\_\_\_\_\_, %al

mov %al, \_buffer(%ebx)\_\_\_\_

\_inc %ecx\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

next\_char:

\_inc %ebx\_\_\_\_\_\_\_\_\_\_\_\_

jmp loop\_begin

loop\_over:

push %ecx

pushl $buffer

pushl $format

call printf

mov $1, %eax # 程序正常退出

mov $0, %ebx

int $0x80

|  |  |
| --- | --- |
| **分 数** |  |
| **评卷人** |  |

1. **问答题（20分）**
2. 可重定位目标文件中有哪些节？各节中主要有什么信息？什么是符号解析？链接的过程是什么？（10分）

可重定位目标文件中有：

.text节：编译汇编后的代码

.data节：已初始化的全局变量、静态局部变量

.rodata节：只读数据

.bss节：未初始化的全局变量、静态局部变量；初始化为0的全局变量、静态局部变量。

此外还有文件头、节头表。

符号解析，是将每个模块中引用的符号，与某个目标模块中的定义符号建立关联。

链接的过程包括符号解析和重定位两方面。符号解析如上所述。重定位则是分别合并代码和数据，并根据代码和数据在虚拟地址空间中的位置，确定每个符号的最终存储地址，然后根据符号的确切地址来修改符号的引用处的地址。

1. 在一个程序的运行过程中（如正在执行一个二维数组求累加和），用户按了键盘上的某个键，计算机系统会做出哪些响应（即一系列的处理过程）？中断分哪几类？请举例说明各类中断在何种情况下产生。（10分）

实模式下，键盘产生一个**中断请求**。

系统中断逻辑电路，将中断请求转为**中断号**，发送给CPU。

CPU响应中断请求。首先保存正在执行程序的当前eip和其他寄存器的内容；然后根据中断号**查询中断向量表**，获取该中断相应的中断处理子程序的入口地址；然后中断当前程序，**跳转到该中断处理子程序中**，处理键盘输入；执行完中断处理子程序后，取回保存的eip和其他寄存器，**返回到原程序**中继续执行。

中断类型包括：

不可屏蔽外部中断：例如电源掉电。

可屏蔽外部中断：例如外部设备产生的中断请求。

CPU检测的内部中断：例如除法出错。

程序检测的内部中断：例如程序中的软中断调用。