******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(A卷)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **考试方式** | **闭卷** | **考试日期** | **2023-12-03** | **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 | 30 | 20 | 10 | 20 | 100 |  |
| **得分** |  |  |  |  |  |  |  |

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

1. **计算机工作基本原理填空（共20分）**

**1、在对某程序进行调试时，看到如下一段信息（10分）：**

**004116DA 83 7D FC 00 cmpl ………………**

**004116DE 75 08**  **jne 004116E8**

**004116E0 8B 45 FC mov ………………**

**004116E3 89 45 F8 mov ………………**

**004116E6 EB 07 jmp 004116EF**

**004116E8 C7 45 F8 01 00 00 00 movl ………………**

**004116EF E8 2D FB FF FF callq func (00411221) ; func为函数名**

**004116F4 C7 45 F4 21 12 41 00 movl $0x411221, -0x0c(%ebp)**

**004116FB FF 55 F4 callq -0x0c(%ebp)**

………… ………… …………

① 观察指令在内存中的存放形式(每个空对应一个16进制字节数据) （2分）：

004116DA \_\_\_ \_\_\_\_ \_\_\_ \_\_\_ \_\_\_ \_\_\_ \_\_\_\_\_ \_\_\_\_\_

004116E2 ………………………………

② 设当前 eip 为 0x004116DE，在取出eip指向的指令并进行译码后，eip = \_\_\_\_\_\_\_；

③ 在0x004116DE 处的指令为： 75 08 jne 004116E8，直观上是在zf =\_\_\_\_\_时，会将0x004116E8 →eip , 计算出该地址值的方法 ；当转移条件不成立时，则该指令执行完成， eip （会、不会）改变。

④ 004116EF 处的指令中有0xFFFFFB2D，计算出该值的方法是 ；

⑤ 004116FB 处的子程序调用指令，调用的子程序的入口地址是 ，执行该处语句时，CPU会将 0x 压入堆栈中。

⑥ 执行子程序中的RET指令时，CPU会 →eip。

2、设一个C语言程序中定义有全局数组 int array[5]; array的起始地址（即&array[0]）为 0x420100，每个元素占4个字节，现要将数组的第2个元素(即array[2])置为20。按指定的寻址方式写出实现该功能的机器指令段（汇编语句段）。（10分，每空1分）

注：完成该功能的C语句有：“array[2]=20; \*(array +2) =20; int i=2; array[i]=20; int \*p=array; p[i] = 20;”等等。不同的C语句写法编译后，生成的机器指令不同，会出现多种寻址方式访问同一单元。

① 使用直接寻址方式，机器指令中含有操作数的偏移地址 ：

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

② 使用寄存器间接寻址方式，先将操作数的地址送ebx，再访存：

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ； 给 ebx赋值

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

③ 使用变址寻址方式，先将元素的下标（即2）送eax，再访存：

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_； 给 eax赋值

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

④ 使用基址加变址寻址方式，先将数组的起始地址送eax，第2个元素在数组中的偏移字节数送ebx，再访存：

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ； 给 eax赋值

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ； 给 ebx赋值

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

⑤ 对于全局变量，其空间分配在\_\_\_\_\_\_\_\_\_。而对于非静态的局部变量，其空间分配在\_\_\_\_\_\_\_\_，对应的地址表达方式一般为 disp(%ebp) 、disp(%esp) 等。

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

1. **数据存储及C语句转换填空（共30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 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"; // '1'的ASCII是 0x31, 'a'的ASCII是 0x61

result = fadd(x, y);

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

}

调试时，设变量 x 的地址（即&x）为 0xffffd508； y 的地址(即&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，以字节为单位观察内存内容（用16进制数的形式填空，最左边是内存地址）。（6分）

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

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

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

1. 数据传送指令解读 （6分，每空 2分）

“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\_\_\_\_\_\_\_\_\_ 。

1. 函数调用语句解读 （10分，每空2分）

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。

|  |  |
| --- | --- |
|  | 0xffffd4ec |
|  | 0xffffd4f0 |
|  | 0xffffd4f4 |
|  | 0xffffd4f8 |
|  | 0xffffd4fc |
| XXXXXXXX | 0xffffd500 |

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（6分）；

② 刚进入函数fadd内部时，esp = 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_；

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

4. 函数fadd的指令解读 (8分，每小题2分)

函数体对应的反汇编代码有：

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。

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

1. **程序阅读与理解 （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 = 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： \_\_\_\_\_\_\_\_\_\_\_\_ us, %eax 、 mov %eax, ui

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

**⑶ 执行完 ③** 处语句后，

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

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

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

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

**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

**2、阅读下面的程序，回答问题 （10分）。**

**.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？（2分）
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？（3分）
3. 若将 **②** 处的语句改为 “jb lp\_2”,程序运行的结果是什么？(2分)
4. 若漏写了 ③ 处的语句 ，程序运行会出现什么现象？为什么？（3分）

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

1. **程序优化 （共10分）。**
2. 举例说明编写C程序或者编译器优化时利用相应原则进行优化的做法（包括优化前的方法，优化后的方法）。 (10 分，每小题 2分)

① 提高CPU中cache的命中率

② 提高CPU中指令流水线的利用率

③ 使用CPU中处理速度更快的指令

④ 使用CPU中单指令多数据流指令或串操作指令

⑤ 使用多核CPU中多线程处理能力

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

1. **链接和异常控制流问答（20分）**
2. 设一个函数中有语句 int temp=global; 其中global是一个初值为35的int 类型全局变量。编译器对global的定义(int global=35;)和temp=global编译时，分别会在可重定位目标文件中的哪些节生成哪些信息？（10分）
3. 什么是中断和异常？两者有何差别？什么是中断描述符表？中断和异常的响应过程是什么？（10分）