******华中科技大学计算机科学与技术学院2024~2025第一学期**

**“ 计算机系统基础 ”考试试卷 (B卷)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **考试方式** | **闭卷** | **考试日期** | **2025-03-09** | **考试时长** | **150 分钟** |
| **专业班级** |  | **学 号** |  | **姓 名** |  |

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

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

**一、（20分） 数据表示和访问**

1. 设有如下程序，调试时在函数main()结束前观察了变量的地址以及内存单元内容。请将程序、注释、以及内存单元中的内容补充完整。（10分）。

int main( ) {

char s[10] = { "A123" }; // s的地址为 0xffffd4a0

int i = 0x678; // i的地址为 0xffffd4ac

float f = 3.5f; // f的地址为 0xffffd4b0

char \*p = s + 2; // p的地址为 0xffffd4b4

strcpy(p, "\_\_\_\_\_\_\_\_"); // 双引号内的空应填为：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

printf("%s", s); // 屏幕显示：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

printf("%x", \*(int \*)(s + 1)); // 屏幕显示：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

return 0;

}

内存的内容以16进制数的形式显示，最左边是内存地址，每空都需要填一个字节（2个16进制位）。

0xffffd4a0 \_\_\_\_\_ \_\_\_\_\_ 39 38 37 36 35 36

0xffffd4a8 42 00 XX XX \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_

0xffffd4b0 \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_

提示：'A'的ASCII码为0x41, '1'的ASCII码为0x31, 小端存储方式。

浮点数float的编码规则：最高位（第31位）为符号位，23-30位为用移码表示的指数（1、-1的移码值分别为0x7F+1、0x7F-1），0-22位为尾数位。

1. 根据调试时观察到的信息，填写空行处的C语句或将指令语句补充完整（10分）。

int g[5];

int main( ) {

**… …**

int x[5];

int i = 0;

0x000011ad <+32>: movl $0x0, (%esp)

int \*p = 0;

0x000011b4 <+39>: movl $0x0, 0x4(%esp)

g[1] = 0;

0x000011bc <+47>: movl \_\_\_\_\_\_, 0x38(%eax)

x[0] = 10;

0x000011c6 <+57>: \_\_\_\_\_\_ $0xa, 0x8(%esp)

i += 2;

0x000011ce <+65>: addl $0x2, \_\_\_\_\_\_

g[ \_\_\_ ] = \_\_\_\_\_;（C语句）

0x000011d2 <+69>: mov (%esp), %edx

0x000011d5 <+72>: mov (%esp), %ecx

0x000011d8 <+75>: mov %ecx, 0x34(%eax, %edx, 4)

\*(g + \_\_\_\_\_ ) = 10;（C语句）

0x000011df <+82>: movl $0xa, 0x3c(%eax)

p = &x[1];

0x000011e9 <+92>: lea 0x\_\_\_(%esp), %eax

0x000011ed <+96>: add $0x4, %eax

0x000011f0 <+99>: mov %eax, \_\_\_\_\_\_\_\_\_\_

p = \_\_\_\_\_\_\_\_\_\_\_\_;（C语句）

0x000011f4 <+103>: addl $0x8, 0x4(%esp)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;（C语句）

0x000011f9 <+108>: mov 0x4(%esp), %eax

0x000011fd <+112>: movl $0x1, (%eax)

**… …**

}

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

**二、（20分） C语言程序理解及函数调用的机器级实现**

1、阅读下面的C语言程序，回答问题（10分，每空1分）

int main( ) {

short s = 0x1111;

unsigned short us = 0xffff

unsigned int ui;

int i;

int x = 0, y = 0;

ui = us; // ui = 0x \_\_\_\_\_\_\_ （4个字节的16进制数）

i = us; // i = 0x \_\_\_\_\_\_\_\_ （4个字节的16进制数）

ui = s; // ui = 0x \_\_\_\_\_\_\_ （4个字节的16进制数）

i = s; // i = 0x \_\_\_\_\_\_\_\_ （4个字节的16进制数）

if (ui > 0) x = 1; // x = 0x \_\_\_\_\_\_\_ （4个字节的16进制数）

if (i > 0) y = 1; // y = 0x \_\_\_\_\_\_\_ （4个字节的16进制数）

ui = ui >> 1; // ui = 0x \_\_\_\_\_\_\_ （4个字节的16进制数）

i = i >> 1; // i = 0x \_\_\_\_\_\_\_\_ （4个字节的16进制数）

ui = ~ui; // ui = 0x \_\_\_\_\_\_\_ （4个字节的16进制数）

i = -i; // i = 0x \_\_\_\_\_\_\_\_ （4个字节的16进制数）

}

(1) 执行完每条指令后，在相应的位置填写执行结果。

(2) 写出下面指令对应的汇编指令（AT&T格式）为：

ui = us; 汇编指令： \_\_\_\_\_\_ us, %eax、 mov %eax, ui

i = us; 汇编指令： \_\_\_\_\_\_ us, %eax、 mov %eax, i

ui = s; 汇编指令： \_\_\_\_\_\_ s, %eax、 mov %eax, ui

i = s; 汇编指令： movsx s, %ebx、 mov \_\_\_\_\_\_, i

if (ui>0) x=1; 汇编指令： cmp $0, ui、 \_\_\_\_\_\_ L1、 mov $1, x、 L1: …

if (i >0) y=1; 汇编指令： cmp $0, i、 \_\_\_\_\_\_ L2、 mov $1, y、 L2: …

ui = ui >> 1; 汇编指令： \_\_\_\_\_\_ $1, ui

i = i >> 1; 汇编指令： \_\_\_\_\_\_\_\_\_\_, i

ui = ~ui; 汇编指令： \_\_\_\_\_\_ ui

i = -i; 汇编指令： \_\_\_\_\_\_ i

2、函数调用的机器级实现 （10分）

下面给出了子程序sum()和主程序main()的反汇编代码（最左边的是机器指令的地址）。

int sum(int x, int y)

{

0x0000117d <+0>: 55 push %ebp

0x0000117e <+1>: 89 e5 mov %esp, %ebp

0x00001180 <+3>: 83 ec 10 sub $0x10, %esp

0x00001183 <+6>: e8 49 00 00 00 call 0x11d1 <\_\_x86.get\_pc\_thunk.ax>

0x00001188 <+11>: 05 54 2e 00 00 add $0x2e54, %eax

int z = x + y;

0x0000118d <+16>: 8b 55 08 mov 0x8(%ebp), %edx

0x00001190 <+19>: 8b 45 0c mov 0xc(%ebp), %eax

0x00001193 <+22>: 01 d0 add %edx, %eax

0x00001195 <+24>: 89 45 fc mov %eax, -0x4(%ebp)

return z;

0x00001198 <+27>: 8b 45 fc mov -0x4(%ebp), %eax

}

0x0000119b <+30>: c9 leave

0x0000119c <+31>: c3 ret

int main( ) {

… …

int x = 100;

0x000011ad <+16>: c7 45 f4 64 00 00 00 movl $0x64, -0xc(%ebp)

int y = 200;

0x000011b4 <+23>: c7 45 f8 c8 00 00 00 movl $0xc8, -0x8(%ebp)

int z = sum(x, y);

0x000011bb <+30>: ff 75 f8 push -0x8(%ebp)

0x000011be <+33>: ff 75 f4 push -0xc(%ebp)

0x000011c1 <+36>: e8 b7 ff ff ff call 0x117d <sum>

0x000011c6 <+41>: 83 c4 08 add $0x8, %esp

0x000011c9 <+44>: 89 45 fc mov %eax, -0x4(%ebp)

… …

}

|  |  |
| --- | --- |
|  | 0xffffd4e4 |
|  | 0xffffd4e8 |
|  | 0xffffd4ec |
|  | 0xffffd4f0 |
|  | 0xffffd4f4 |
| XXXXXXX | 0xffffd4f8 |

假设执行main()的语句“int z = sum(x, y);”之前，esp值为0xffffd4f8，ebp值为0xffffd508。单步执行“call sum”指令进入sum函数后，回答下面的问题：

(1) 执行到sum函数的 leave指令时（该指令还没有执行），填写右边表格中的相应内存单元中的值。（5分）

(2) 函数参数 x的地址（即&x）是 0x\_\_\_\_\_\_\_\_\_\_\_\_\_。（1分）

(3) 局部变量 z的地址（即&z）是 0x\_\_\_\_\_\_\_\_\_\_\_\_\_。（1分）

(4) 执行“add %edx, %eax” 后， CF=\_\_\_\_, SF=\_\_\_\_\_, ZF=\_\_\_\_\_, OF= \_\_\_\_\_\_\_。（2分）

(5) 执行 “ret ” 后 ，%esp = 0x \_\_\_\_\_\_\_\_\_\_\_\_\_。（1分）

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

1. （20分）计算机系统的基本原理

1、下面显示了C源程序及相应的反汇编指令，分析每条指令并填空。（10分）

extern int f (int x);

extern int g(int x);

main:

{

**… …**

0x000011c3 <+10>: 55 push %ebp

0x000011c4 <+11>: 89 e5 mov %esp, %ebp

0x000011c6 <+13>: 51 push %ecx

0x000011c7 <+14>: 83 ec 24 **sub** $0x24, %esp

**// (1)** sub指令的作用是 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**… …**

int x = 100;

0x000011e0 <+39>: c7 45 e4 64 00 00 00 movl $0x64, -0x1c(%ebp)

int (\*pf[2])(int) = { f, g };

0x000011e7 <+46>: 8d 90 b5 d1 ff ff lea -0x2e4b(%eax), **%edx**

**// (2)** 给出edx值的表达式 \_\_\_\_\_\_\_\_\_\_\_\_

0x000011ed <+52>: 89 55 ec mov %edx, -0x14(%ebp)

**// (3)** 目的操作数是存贮单元还是寄存器 \_\_\_\_\_\_\_\_\_\_\_\_

0x000011f0 <+55>: 8d 80 cb d1 ff ff lea -0x2e35(%eax), %eax

0x000011f6 <+61>: 89 45 f0 mov %eax, -0x10(%ebp)

f(x);

0x000011f9 <+64>: ff 75 e4 push -0x1c(%ebp)

0x000011fc <+67>: e8 8c ff ff ff call 0x118d <f>

// **(4)** 执行call 指令时，会将\_\_\_\_\_\_\_\_\_， 将\_\_\_\_\_\_\_\_\_\_→ eip

// **(5)** 执行函数f()中的返回指令ret时，CPU会\_\_\_\_\_\_\_\_\_\_ → eip。

0x00001201 <+72>: 83 c4 04 add $0x4, %esp

for(int k = 0; k < 2; k++)

0x00001204 <+75>: c7 45 e8 00 00 00 00 movl $0x0, -0x18(%ebp)

0x0000120b <+82>: eb **19** jmp **0x1226 <main+109>**

**// (6)** 在取出该指令并进行译码后，eip = 0x\_\_\_\_\_\_\_\_\_\_\_\_

// **(7)** 机器码19代表的意思是 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x00001222 <+105>: 83 45 e8 01 addl $0x1, -0x18(%ebp)

**0x00001226 <+109>:** 83 7d e8 01 cmpl $0x1, -0x18(%ebp)

0x0000122a <+113>: 7e e1 jle 0x120d <main+84>

{

x = pf[k](x);

0x0000120d <+84>: 8b 45 e8 mov -0x18(%ebp), %eax

0x00001210 <+87>: 8b 44 85 ec mov -0x14(%ebp, %eax,4), %eax

0x00001214 <+91>: 83 ec 0c sub $0xc, %esp

0x00001217 <+94>: ff 75 e4 push -0x1c(%ebp)

0x0000121a <+97>: ff d0 call \*%eax

**// (8)** 执行call指令时, 采用 \_\_\_\_\_\_\_\_\_\_ 寻址方式, 得到函数的入口地址

0x0000121c <+99>: 83 c4 10 add $0x10, %esp

0x0000121f <+102>: 89 45 e4 mov %eax, -0x1c(%ebp)

}

return x;

0x0000122c <+115>: 8b 45 e4 mov -0x1c(%ebp), %eax

**// (9)** eax值的作用是 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2、二进制拆弹：主程序main()的部分源码如下，利用函数read\_line()从键盘输入一个字符串到缓冲区input，函数phase\_1()比较input和它内部的一个字符串是否相同，若相同则表示拆弹成功并正常返回，否则（拆除失败）报错并退出程序。函数phase\_1()没有源码，下面列出了通过gdb调试反汇编得到的汇编指令。请描述拆弹过程。（10分）

**int main() {**

**… …**

printf("Gate 1: input a string that meets the requirements. \n");

input = read\_line(); //输入一行信息

phase\_1(input); //第一关炸弹, 若拆除失败则报错并退出程序

printf("Phase 1 passed! \n"); //拆弹成功

**… …**

}

**phase\_1 {**

0x555555555991 <phase\_l> endbr64

0x555555555995 <phase\_1+4> push %rbp

0x555555555996 <phase\_1+5> mov %rsp, %rbp

0x555555555999 <phase\_1+8> sub 0x20, %rsp

0x55555555599d <phase\_1+12> mov %rdi, -0x18(%rbp)

0x5555555559al <phase\_1+16> movzbl 0x2af9 (%rip), %eax #0x5555555584al <studentid+9>

0x5555555559a8 <phase\_1+23> sub $0x30, %eax

0x5555555559ab <phase\_1+26> mov %al, -0x5(%rbp)

0x5555555559ae <phase\_1+29> movsbl -0x5(%rbp), %eax

0x5555555559b2 <phase\_1+33> mov %eax, -0x4(%rbp)

0x5555555559b5 <phase\_1+36> mov -0x4(%rbp), %edx

0x5555555559b8 <phase\_1+39> mov %rdx, %rax

0x5555555559bb <phase\_1+42> shl $0x2, %rax

0x5555555559bf <phase\_1+46> add %rdx, %rax

0x5555555559c2 <phase\_1+49> lea 0x0(,%rax,4), %rdx

0x5555555559ca <phase\_1+57> add %rdx,%rax)

0x5555555559cd <phase\_1+60> add %rax, %rax

0x5555555559d0 <phase\_1+63> lea 0x2649(%rip), %rdx #0x555555558020 <special>

0x5555555559d7 <phase\_1+70> add %rax, %rdx

0x5555555559da <phase\_1+73> mov -0x18(%rbp), %rax

0x5555555559de <phase\_1+77> mov %rdx, %rsi

0x5555555559el <phase\_1+80> mov %rax, %rdi

0x5555555559e4 <phase\_1+83> call 0x555555555698 <strings\_not\_equal>

0x5555555559e9 <phase\_1+88> test %eax, %eax

0x5555555559eb <phase\_1+90> je 0x5555555559f2 <phase\_1+97>

0x5555555559ed <phase\_ 1+92> call 0x555555555970 <explode\_bomb>

0x5555555559f2 <phase\_1+97> nop

0x5555555559f3 <phase\_1+98> leave

0x5555555559f4 <phase\_1+99> ret

}

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

1. （20分）程序编译及优化

1、充分利用Cache可以显著提高程序的执行效率，举例说明如何提高Cache的命中率？（5分）

2、如何利用CPU的指令流水线提高程序的执行效率，请给出3种不同类型的方案。（6分）

3、除了利用CPU的流水线和Cache提高程序的执行效率外，还可以利用CPU的其他特性提高程序执行效率。请给出5种不同类型的利用CPU其他特性的优化示例。（5分）

4、指出下述C语言程序段执行效率不高的原因。（4分）

char s[100000];

int c = 0;

for(int k = 0; k < strlen(s); k++) {

if(s[k] == ’A’) c++;

}

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

1. （20分）链接和异常控制流

**1、设有如下程序（10分）**

#include <stdio.h>

extern int f(int);

int g = 100;

int (\*pf)(int) = f;

int main()

{ int x;

const char \*s;

s = “Hello”;

x = pf(g);

x = x + g;

printf("%s %d \n", s, x);

return 0;

}

在编译生成的可重定位目标文件中，会有数据节（.data）、数据重定位节（.rel.data）、只读数据节（.rodata）、代码节（.text）、代码重定位节（.rel.text）、符号表节（.symtab）、字符串节（.strtab）等。回答下面的问题。

(1) 对于上述的7个节，说明符号g和pf的哪些信息分别存贮在哪些节中？（6分）

(2) 哪些C语句中的哪些信息需要重定位？（不用给出具体的机器指令）？（4分）

**2、异常控制流 （10分）**

(1) 中断和异常的区别是什么？（2分）

(2) 异常有哪些类型？各种类型的异常处理程序执行完后返回到哪里？（3分）

(3) 简述键盘按键中断的响应过程。（3分）

(4) 中断描述符表的作用是什么？（2分）